```
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDD          DDD RRR          RRR       III      VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III      VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III      VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III      VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III      VVV         VVV  EEE              RRR          RRR
DDD          DDD RRRRRRRRRRRR           III      VVV         VVV  EEEEEEEEEEEE     RRRRRRRRRRRR
DDD          DDD RRRRRRRRRRRR           III      VVV         VVV  EEEEEEEEEEEE     RRRRRRRRRRRR
DDD          DDD RRRRRRRRRRRR           III      VVV         VVV  EEEEEEEEEEEE     RRRRRRRRRRRR
DDD          DDD RRR   RRR              III      VVV         VVV  EEE              RRR   RRR
DDD          DDD RRR   RRR              III      VVV         VVV  EEE              RRR   RRR
DDD          DDD RRR   RRR              III      VVV         VVV  EEE              RRR   RRR
DDD          DDD RRR       RRR          III        VVV     VVV    EEE              RRR       RRR
DDD          DDD RRR       RRR          III        VVV     VVV    EEE              RRR       RRR
DDD          DDD RRR       RRR          III        VVV     VVV    EEE              RRR       RRR
DDDDDDDDDDDD     RRR          RRR    IIIIIIIII       VVV         EEEEEEEEEEEEEEE  RRR          RRR
DDDDDDDDDDDD     RRR          RRR    IIIIIIIII       VVV         EEEEEEEEEEEEEEE  RRR          RRR
DDDDDDDDDDDD     RRR          RRR    IIIIIIIII       VVV         EEEEEEEEEEEEEEE  RRR          RRR
```

```
XX      XX    QQQQQQ    DDDDDDDD    RRRRRRRR     IIIIII    VV      VV  EEEEEEEEEE  RRRRRRRR
XX      XX    QQQQQQ    DDDDDDDD    RRRRRRRR     IIIIII    VV      VV  EEEEEEEEEE  RRRRRRRR
XX      XX  QQ      QQ  DD      DD  RR      RR     II      VV      VV  EE          RR      RR
XX      XX  QQ      QQ  DD      DD  RR      RR     II      VV      VV  EE          RR      RR
  XX  XX    QQ      QQ  DD      DD  RR      RR     II      VV      VV  EE          RR      RR
  XX  XX    QQ      QQ  DD      DD  RR      RR     II      VV      VV  EE          RR      RR
    XX      QQ      QQ  DD      DD  RRRRRRRR       II      VV      VV  EEEEEEEE    RRRRRRRR
    XX      QQ      QQ  DD      DD  RRRRRRRR       II      VV      VV  EEEEEEEE    RRRRRRRR
  XX  XX    QQ  QQ  QQ  DD      DD  RR  RR         II      VV      VV  EE          RR  RR
  XX  XX    QQ  QQ  QQ  DD      DD  RR  RR         II      VV      VV  EE          RR  RR
XX      XX  QQ    QQ    DD      DD  RR    RR       II        VV  VV    EE          RR    RR      ....
XX      XX  QQ    QQ    DD      DD  RR    RR       II        VV  VV    EE          RR    RR      ....
XX      XX    QQQQ  QQ  DDDDDDDD    RR      RR   IIIIII        VV      EEEEEEEEEE  RR      RR    ....
XX      XX    QQQQ  QQ  DDDDDDDD    RR      RR   IIIIII        VV      EEEEEEEEEE  RR      RR    ....

LL              IIIIII      SSSSSSSS
LL              IIIIII    SSSSSSSS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II        SSSSSS
LL                II        SSSSSS
LL                II              SS
LL                II              SS
LL                II              SS
LL                II              SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

Table of contents

(9) 1121 Standard tables
(10) 1204 Local driver storage
(11) 1439 CONTROL_INIT - INITIALIZE DEQNA DEVICE
(12) 1465 CLONED_UCB - INITIALIZE THE CLONED UCB
(13) 1495 UNIT_INIT - INITIALIZE THE DEQNA UNIT
(14) 1598 FFI_INIT - FFI INTERFACE INITIALIZATION ROUTINE
(15) 1637 XMT_FDT - TRANSMIT I/O OPERATION FDT ROUTINE
(16) 1753 XMT_FFI_START - START FAST INTERFACE TRANSMIT OPERATION
(17) 1821 XMT_START - START TRANSMIT OPERATION
(18) 2196 RCV_FDT - RECEIVE I/O OPERATION FDT ROUTINE
(19) 2276 RCV_START - START RECEIVE I/O OPERATION
(20) 2357 SUBROUTINES TO FIND SHR DATA STRUCTURE
(21) 2419 ALT_START - ALTERNATE START I/O ROUTINE
(22) 2490 SETMODE_FDT - SET MODE I/O OPERATION FDT DISPATCH ROUTINE
(24) 2920 SHR_UCB - CREATE SHARED UCB
(25) 3061 CHECK_PARAM - CHECK SHARED USERS PARAMETERS
(26) 3123 SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
(27) 3155 SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
(28) 3276 READ_LINE_CTR - READ THE LINE COUNTERS
(28) 3277 READ_CIRC_CTR - READ THE CIRCUIT COUNTERS
(29) 3387 GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
(30) 3443 CHECK_BUFS - CHECK P1 AND P2 BUFFERS FOR WRITE ACCESS
(31) 3493 CHECK_P1 - CHECK P1 BUFFER ADDRESS FOR WRITE ACCESS
(32) 3532 ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
(33) 3599 STARTIO - START I/O OPERATION
(34) 3689 START - START UNIT'S PROTOCOL
(35) 4215 SETUP_MODE - SETUP THE TRANSMIT BUFFER TO INIT QNA
(36) 4329 FILLRCVLIST - FILL RECEIVE BUFFER LIST
(37) 4408 START_RECEIVE - START ANY RECEIVE REQUESTS PENDING
(38) 4564 LOAD_PORT - LOAD CSR'S WITH COMMAND REQUEST
(39) 4647 QNA_INTR - QNA INTERRUPT SERVICE ROUTINE
(40) 4709 SCHED_FORK - SCHEDULE THE FORK PROCESS
(40) 4710 SCHED_FORKC - SCHEDULE THE FORK PROCESS WITH R3 CLEAR
(41) 4754 FORK_PROC - Error and completion fork process handling
(42) 5054 RCV_ERROR - Process receive errors
(42) 5055 XMT_ERROR - Process transmit errors
(43) 5122 SUBROUTINES TO FIND SHR MATCH ON SOURCE ADDRESS
(44) 5184 COPY_RCV - Copy a receive buffer for the PROMISCUOUS user
(45) 5256 FINISH_XMT_FFI - Finish FAST interface transmit processing
(46) 5288 FINISH_RCV_FFI - Finish FAST receive processing
(47) 5334 FINISH_RCV_IO - Finish receive I/O processing
(48) 5473 ASSEM_PKTS - Assemble receive packets
(49) 5648 MOP_CTR_REQUEST - PROCESS MOP READ COUNTERS REQUEST
(50) 5693 MOP_CTR_BUILD - BUILD THE MOP COUNTER RETURN MESSAGE
(51) 5761 REG_DUMP - DEQNA ERROR LOG AND DIAGNOSTICS REGISTER DUMP
(52) 5795 RESTART_ROUT - PROCESS EXPIRATION OF RESTART TIMER
(53) 5885 TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
(54) 5962 TIMEOUT - TIMEOUT SERVICE ROUTINE
(55) 6006 ALLOC_CDB - ALLOCATE THE CDB
(56) 6072 SHUTDOWN_QNA - SHUTDOWN QNA AND ALL UNITS
(57) 6227 SHUTDOWN - SHUT DOWN UNIT
(57) 6228 SHUTDOWN_PROTYP - SHUT DOWN PROTOCOL TYPE
(58) 6497 BLD_STOP_IRP - Build an IRP to reset promiscuous mode
(59) 6544 BLD_STRT_IRP - Build a point-to-point startup IRP
(60) 6588 BLD_IRP - Build an IRP routine
(61) 6637 CLEANUP_SHR - CLEANUP ALL I/O ON SHARE DATA STRUCTURE
(61) 6638 DELETE_SHR - DELETE SHR DATA STRUCTURE
(62) 6766 CANCEL - CANCEL I/O ON UNIT
(63) 6965 SUBROUTINES TO FIND SHR DATA STRUCTURE GIVEN PCB AND CHAN
(64) 7022 FIND_POINT_UCB - Find the point to point UCB

```
(65)    7111        ADD_MULTI - ADD UP ALL THE MULTICAST ADDRESSES
(66)    7182        MOVE_MULTI - COPY THE MULTICAST ADDRESS LIST
(67)    7212        ROUTINES TO SAVE/RESTORE UCB'S MULTICAST ADDRESS LIST
(68)    7248        VALIDATE_P2 - VALIDATE P2 BUFFER PARAMETERS
(69)    7417        CHANGE_PARAM - UPDATE UCB/CDB BASED ON P2 BUFFER PARAMETERS
(70)    7546        RETURN_P2,  Return UCB/CDB buffer parameters
(71)    7654        VALID_MULTI - VALIDATE THE MULTICAST ADDRESS LIST
(72)    7711        VALID_PHYAD - VALIDATE THE PHYSICAL ADDRESS
(73)    7773        SET_MULTI - SET THE UCB MULTICAST ADDRESS LIST
(74)    7927        SET_PHYAD - SET THE PHYSICAL ADDRESS
(74)    7928        SET_DESAD - SET THE DESTINATION ADDRESS
(75)    8009        RETURN_MULTI - RETURN THE MULTICAST ADDRESS LIST
(76)    8052        MATCH_MULTI - CHECK MULTICAST ADDRESS
(77)    8097        MATCH_ADDRESS - FIND A MATCH ON A MULTICAST ADDRESS
(78)    8139        POKE_USER - DELIVER ATTENTION ASTS
(79)    8181        MATCH_PROTYP - Match protocol type
(79)    8182        MATCH_PROMTYP - Find the promiscuous user
```

```
0000     1          .TITLE  XQDRIVER - VAX/VMS QNA driver
0000     2          .IDENT  'V04-000'
0000     3  ;
0000     4  ;*****************************************************************************
0000     5  ;*                                                                           *
0000     6  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     7  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     8  ;*  ALL RIGHTS RESERVED.                                                     *
0000     9  ;*                                                                           *
0000    10  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED     *
0000    11  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
0000    12  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
0000    13  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY     *
0000    14  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY     *
0000    15  ;*  TRANSFERRED.                                                             *
0000    16  ;*                                                                           *
0000    17  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE     *
0000    18  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    19  ;*  CORPORATION.                                                             *
0000    20  ;*                                                                           *
0000    21  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS     *
0000    22  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    23  ;*                                                                           *
0000    24  ;*                                                                           *
0000    25  ;*****************************************************************************
0000    26  ;++
0000    27  ; FACILITY:
0000    28  ;
0000    29  ;       VAX/VMS DEQNA QUEUE I/O DRIVER
0000    30  ;
0000    31  ; ABSTRACT:
0000    32  ;
0000    33  ;       This module contains the DEQNA driver FDT routines,
0000    34  ;       interrupt dispatcher, interrupt service and fork routines.
0000    35  ;
0000    36  ; AUTHOR:
0000    37  ;
0000    38  ;       Rod Gamache     26-Jul-1983
0000    39  ;
0000    40  ;
0000    41  ; MODIFICATION HISTORY:
0000    42  ;
0000    43  ;       V03-013 RNG0013         Rod Gamache             23-Jul-1984
0000    44  ;               Fix mode setup and shutdown.
0000    45  ;               Change default on ORB$B_FLAGS to not set ACL queue present bit.
0000    46  ;
0000    47  ;       V03-012 RNG0012         Rod Gamache             6-Jul-1984
0000    48  ;               Fix ALLOC_CDB to intialize the address of UCB unit 0.
0000    49  ;               Fix problems with re-starting FFI users and deleting
0000    50  ;               transmits on error.
0000    51  ;               Fix bug when disabling PROMiscuous mode in hardware.
0000    52  ;               Fix MOP read counters request.
0000    53  ;
0000    54  ;       V03-011 RNG0011         Rod Gamache             17-May-1984
0000    55  ;               Account for 4 bytes of CRC on received messages, when
0000    56  ;               returning byte count from assemble_pkt.
0000    57  ;               Change the way the "set default" modifier for the
```

```
0000     58 ;                  Physical address is processed.
0000     59 ;
0000     60 ;       V03-010 RNG0010         Rod Gamache             11-May-1984
0000     61 ;                Fix the setup of the multicast address list for all
0000     62 ;                users after the first.
0000     63 ;
0000     64 ;       V03-009 RNG0009         Rod Gamache             4-May-1984
0000     65 ;                Add DEV$M_NET flag to device characteristics.
0000     66 ;
0000     67 ;       V03-008 RNG0008         Rod Gamache             19-Apr-1984
0000     68 ;                Fix call to EXE$ALOPHYCNTG to be absolute addressing.
0000     69 ;
0000     70 ;       V03-007 RNG0007         Rod Gamache             12-Apr-1984
0000     71 ;                Fix problem with incorrect useage of the FFI interface.
0000     72 ;                Also, return the Hardware Ethernet Address.
0000     73 ;
0000     74 ;       V03-006 KPL0001         Peter Lieberwirth       9-Apr-1984
0000     75 ;                Use EXE$ALOPHYCNTG to allocate physically-contiguous IO buffer
0000     76 ;                required by u-VAX I on the QNA.
0000     77 ;
0000     78 ;       V03-005 LMP0221         L. Mark Pilant,         27-Mar-1984  12:02
0000     79 ;                Change UCB$L_OWNUIC to ORB$L_OWNER and UCB$W_VPROT to
0000     80 ;                ORB$W_PROT.
0000     81 ;
0000     82 ;       V03-004 RNG0004         Rod N. Gamache          6-Feb-1984
0000     83 ;                Set the XQ unit to RUN state when the QNA is initialized.
0000     84 ;                Make the SETMODE descriptor use a word for the length, rather
0000     85 ;                than a longword.
0000     86 ;
0000     87 ;       V03-003 TMK0002         Todd M. Katz            03-Feb-1984
0000     88 ;                When the DEQNA times out because of QBUS or device controller
0000     89 ;                power failure, then call back all protocols which are using the
0000     90 ;                FFI interface and have defined an FFI$L_ERROR asynchronous
0000     91 ;                error routine. What this involves is noticing within the
0000     92 ;                routine SHUTDOWN by means of the UCB$V_POWER status bit that a
0000     93 ;                power failure has occurred. In such a circumstance the port
0000     94 ;                driver wants to call the asynchronous error routine of the
0000     95 ;                protocol before doing anything else provided the protocol has
0000     96 ;                initialized the FFI interface, defined an asynchronous error
0000     97 ;                routine, and the UCB for the protocol is both on-line and
0000     98 ;                initialized.
0000     99 ;
0000    100 ;       V03-002 TMK0001         Todd M. Katz            03-Feb-1984
0000    101 ;                Make the following changes to the driver:
0000    102 ;
0000    103 ;                1. I have created a NI device dependent UCB extension within
0000    104 ;                   $UCBDEF. This extension contains definitions for
0000    105 ;                   UCB$L_NI_HWAPTR and UCB$L_NI_MLTPTR, two new locations
0000    106 ;                   to be contained within the UCBs of all NI datalink drivers.
0000    107 ;                   I have therefore modified the XQDRIVER's UCB definition
0000    108 ;                   so that the DEQNA specific UCB fields begin immediately
0000    109 ;                   following the NI device dependent UCB extension.
0000    110 ;
0000    111 ;                   UCB$L_NI_HWAPTR is initialized when the CDB is first
0000    112 ;                   allocated to contain the address of CDB_G_HWA, the CDB
0000    113 ;                   location which contains the NI device's unique hardware
0000    114 ;                   address. UCB$L_NI_MLTPTR is initialized within the unit
```

```
0000   115 ;                   unitialization routine to contain the address of the table
0000   116 ;                   of multicast addresses defined for this protocol type.
0000   117 ;                   Both of these values must be accessible to the NI-SCS port
0000   118 ;                   driver. The NI-SCS port driver has access to the UCBs of the
0000   119 ;                   NI devices participating in SCS clusters, but it doesn't have
0000   120 ;                   initimate knowledge of how the UCBs and CDBs are layed out
0000   121 ;                   for each NI device. This UCB extension provides a means for
0000   122 ;                   the NI-SCS port driver to locate these values without knowing
0000   123 ;                   the exact layout of each of the NI device's UCB and CDB.
0000   124 ;
0000   125 ;           2. Whenever a protocol is to be started up on a DEQNA allow the
0000   126 ;                   initiator of the SETMODE+STARTUP to specify 0 receive
0000   127 ;                   buffers instead of the former minimum of 1. The effect on
0000   128 ;                   this protocol is that it must have a READ outstanding at all
0000   129 ;                   times in order to guarentee that it will receive all
0000   130 ;                   datagrams specifying its protocol. If a datagram was
0000   131 ;                   received for this protocol, and there wasn't a read
0000   132 ;                   outstanding, then because 0 receive buffers can be queued
0000   133 ;                   (or saved) for this protocol, the receive buffer would be
0000   134 ;                   deallocated to pool, and the message it contained lost.
0000   135 ;
0000   136 ;                   This change is extremely useful for those users who are
0000   137 ;                   making use of the FFI interface provided by this port driver.
0000   138 ;                   Between the time the user issues a SETMODE+STARTUP, and the
0000   139 ;                   time the user initializes the FFI interface by calling
0000   140 ;                   FFI_INIT, it would be possible for the port driver to have
0000   141 ;                   received messages with this user's protocol, and to have
0000   142 ;                   queued them up to the appropriate UCB. Now, if the user
0000   143 ;                   never makes use of the QIO or ALTSTART interfaces, but just
0000   144 ;                   uses the FFI interfaces for communication, these messages,
0000   145 ;                   which are probably stale by this time, will never be
0000   146 ;                   received, and the buffers that contain them will effectively
0000   147 ;                   be lost forever. With this change, a user who wishes to do
0000   148 ;                   all his/her communication via the FFI interface can guarentee
0000   149 ;                   that a situation such as this can never arise, and buffers
0000   150 ;                   such as these can never be "lost".
0000   151 ;
0000   152 ;           4. Add the capability of requesting that the physical address
0000   153 ;                   of the DEQNA device be set to the default DECnet address,
0000   154 ;                   when the device is first initialized. This new capability
0000   155 ;                   is requested by means of the NMA$C_LNMCN_SDF mode value
0000   156 ;                   specified within the modifier field of the NMA$C_PCLI_PHA
0000   157 ;                   parameter.
0000   158 ;
0000   159 ;       V03-001 RNG0001         Rod Gamache             08-Dec-1983
0000   160 ;               Add new QNA "hand-shake" to prevent driver from reading
0000   161 ;               incorrect status from hardware.
0000   162 ;
0000   163 ;--
0000   164
0000   165 ;
0000   166 ; EXTERNAL SYMBOLS
0000   167 ;
0000   168         $ABDDEF                         ; Define ABDs
0000   169         $ACBDEF                         ; Define AST control block
0000   170         $CANDEF                         ; Define CANCEL reason codes
0000   171         $CCBDEF                         ; Define CCB offsets
```

```
                        $CRBDEF                         ; Define CRB
                        $CXBDEF                         ; Define CXB
                        $DCDEF                          ; Define device classes and types
                        $DDBDEF                         ; Define DDB
                        $DEVDEF                         ; Define device characteristics
                        $DPTDEF                         ; Define DPT
                        $DYNDEF                         ; Define dynamic data structures
                        $FFIDEF                         ; Fast Interface definitions
                        $FKBDEF                         ; Fork block definitions
                        $IDBDEF                         ; Define IDB
                        $IODEF                          ; Define I/O function codes
                        $IPLDEF                         ; Define IPLs
                        $IRPDEF                         ; Define IRP
                        $JIBDEF                         ; Define JIB
                        $MSGDEF                         ; Define SYSTEM MESSAGES
                        $NMADEF                         ; Define Network Management Codes
                        $ORBDEF                         ; OBJECT'S RIGHTS BLOCK OFFSETS
                        $PCBDEF                         ; Define PCB
                        $PRDEF                          ; Processor register definitions
                        $PRVDEF                         ; Privilege bit definitons
                        $PTEDEF                         ; Define system PTEs
                        $SSDEF                          ; Define System Status Codes
                        $TQEDEF                         ; Define TQE offsets
                        $UBADEF                         ; Define UBA symbols
                        $UCBDEF                         ; Define UCB
                        $VADEF                          ; Define Virtual Address bits
                        $VECDEF                         ; Define CRB VECTOR
                        $XMDEF                          ; Define DECnet datalink characteristics
                ;
                ; Local symbol definitions
                ;

                ;
                ; Define the following symbol to enable use of point-to-point mode
                ;
00000001                POINT = 1                       ; Enable use of point-to-point mode

                ;
                ; Argument list offsets for QIO
                ;
00000000        P1      = 0                             ; Parameter 1
00000004        P2      = 4                             ; Parameter 2
00000008        P3      = 8                             ; Parameter 3
0000000C        P4      = 12                            ; Parameter 4
00000010        P5      = 16                            ; Destination/Source address

                ;
                ; Constants
                ;
                $EQU    BRDCST1         <^XFFFFFFFF>    ; Broadcast address
                $EQU    BRDCST2         <^XFFFF>        ;       ...
                $EQU    MAX_C_MLT       12              ; Maximum number of multicast addresses
                $EQU    MAX_C_XMT       4               ; Maximum number of entries in XMT ring
                $EQU    MAX_C_RCV       8               ; Maximum number of entries in RCV ring
                $EQU    MAX_C_XMTUV1    1               ; Maximum number of XMTs on micro-VAX I
                $EQU    MAX_C_RCVUV1    5               ; Maximum number of RCVs on micro-VAX I
                $EQU    MAX_PRT_SIZE    1500            ; Size of maximum Ethernet user data
```

```
0000    229 $EQU    XQ_C_HEADER     14                  ; Size of Ethernet header
0000    230 $EQU    XQ_C_CRC        4                   ; Size of Ethernet CRC
0000    231 $EQU    XQ_C_CNTSIZ     2                   ; Size of packet count field
0000    232 $EQU    MAX_BUFSIZ_UV1  MAX_PKT_SIZE+XQ_C_HEADER
0000    233 $EQU    UV1_BUFFER_AREA <<MAX_C_XMTUV1+MAX_C_RCVUV1>*MAX_BUFSIZ_UV1>
0000    234 $EQU    MAX_C_CHAIN     1                   ; Maximum number of extra segments in
0000    235                                             ;  a receive buffer chain
0000    236
0000    237 $EQU    NI_CTR_PROTYP   <^X0260>            ; Ethernet read counters protocol 60-02
0000    238 $EQU    NI_CTR_READ     9                   ; Read counters request function
0000    239 $EQU    NI_CTR_REPLY    11                  ; Read counters reply function
0000    240
0000    241 $EQU    INIT_C_QUOTA    6*1500              ; Allow for 6 of the largest buffers
0000    242 $EQU    INIT_C_AQUOTA   2                   ; Allow for 2 additional buffers
0000    243 $EQU    INIT_C_BUFSIZE  128                 ; Size of init (setup mode) buffer
0000    244 $EQU    DSC$A_POINTER   4                   ; Pointer to data in buffer descriptor
0000    245 $EQU    MIN_PRT_SIZE    46                  ; Size of user data in a runt packet
0000    246 $EQU    TQE_C_DELTA     2                   ; 2 second timer interval
0000    247 $EQU    TQE_DELTA       TQE_C_DELTA*10000*1000 ; Delta interval (in 100 nsec)
0000    248 $EQU    RESTART_DELTA   3*10000*1000        ; RESTART interval - 3 Seconds
0000    249 $EQU    XMT_C_TIM       8                   ; XMITS must take less than 8 seconds
0000    250 $EQU    XMT_TIM         <<XMT_C_TIM+2>/2>
0000    251 $EQU    DNI_C_TIM       10                  ; DNI settings must take less than 10s.
0000    252 $EQU    DNI_TIM         <<DNI_C_TIM+2>/2>
0000    253 $EQU    XQ_C_ADDRCV     64                  ; Size to add to received packets
0000    254 $EQU    XQ_C_STPRO      <^X0660>            ;% PT-TO-PT Starting protocol type 60-06
0000    255 $EQU    IPL$_XQ_FIPL    8                   ; Fork IPL
0000    256 $EQU    IPL$_XQ_DIPL    21                  ; Device IPL
0000    257
0000    258 ;
0000    259 ; Local macros
0000    260 ;
0000    261 .MACRO  SETBIT  VAL,FLAG
0000    262         .NTYPE  $$,VAL
0000    263         .IF EQ  <$$-^X0EF>
0000    264         .IF NDF VAL
0000    265         BBSS    S^#VAL,FLAG,.+1
0000    266         .IFF
0000    267         .IF LT <VAL-8>
0000    268         BISB    #<1@VAL>,FLAG
0000    269         .IFF
0000    270         BBSS    #VAL,FLAG,.+1
0000    271         .ENDC
0000    272         .ENDC
0000    273         .IFF
0000    274         BBSS    VAL,FLAG,.+1
0000    275         .ENDC
0000    276 .ENDM   SETBIT
0000    277 ;
0000    278 ; ======================
0000    279 ;
0000    280 .MACRO  CLRBIT  VAL,FLAG
0000    281         .NTYPE  $$,VAL
0000    282         .IF EQ  <$$-^X0EF>
0000    283         .IF NDF VAL
0000    284         BBCC    S^#VAL,FLAG,.+1
0000    285         .IFF
```

```
0000   286          .IF LT <VAL-8>
0000   287          BICB    #<1@VAL>,FLAG
0000   288          .IFF
0000   289          BBCC    #VAL,FLAG,.+1
0000   290          .ENDC
0000   291          .ENDC
0000   292          .IFF
0000   293          BBCC    VAL,FLAG,.+1
0000   294          .ENDC
0000   295 .ENDM   CLRBIT
0000   296 ;
0000   297 ; ======================
0000   298 ;
0000   299 .MACRO  INCC    COUNTER,CONTEXT=L,?L    ; Increment counter
0000   300          INC'CONTEXT     COUNTER         ; Do Increment
0000   301          BCC     L                       ; Br if no carry set
0000   302          DEC'CONTEXT     COUNTER         ; Leave at maximum value
0000   303 L:
0000   304 .ENDM   INCC
0000   305 ;
0000   306 ; ======================
0000   307 ;
0000   308 .MACRO  CNTR    CURCNT,COUNTER,CONTEXT=L,?L  ; Accumlate counter
0000   309          ADD'CONTEXT     CURCNT,COUNTER  ; Do addition
0000   310          BCC     L                       ; Br if no carry set
0000   311          MNEG'CONTEXT    #1,COUNTER      ; Leave at maximum value
0000   312 L:
0000   313 .ENDM   CNTR
0000   314 ;
0000   315 ; ======================
0000   316 ;
0000   317 .MACRO  PUSHQ   ARG                     ; Push a quadword
0000   318          MOVQ    ARG,-(SP)               ; Save argument on stack
0000   319 .ENDM   PUSHQ
0000   320 ;
0000   321 ; ======================
0000   322 ;
0000   323 .MACRO  POPQ    ARG                     ; Pop a quadword
0000   324          MOVQ    (SP)+,ARG               ; Restore argument
0000   325 .ENDM   POPQ
0000   326 ;
0000   327 ; ======================
0000   328 ;
0000   329 .MACRO  PARAM   TYPE,OFFSET,WIDTH,MIN,MAX,INVALID,BASE=UCB,STRING,-
0000   330                      SIZE,CHECK=YES
0000   331 ;
0000   332 ;       Macro to generate the parameter tables
0000   333 ;
0000   334 ; Inputs:
0000   335 ;
0000   336 ;       TYPE = Parameter type
0000   337 ;       OFFSET = Offseter in UCB/CDB to current value
0000   338 ;       WIDTH = Width of field in UCB/CDB (B,W,L)
0000   339 ;       MIN = Minimum value parameter is allowed to take
0000   340 ;       MAX = Maximum value parameter is allowed to take
0000   341 ;       INVALID = Invalid flags in status word
0000   342 ;       BASE = Data base (CDB,UCB)
```

```
0000   343 ;       STRING = Parameter is a string value
0000   344 ;       SIZE = Maximum size of string parameter in bytes
0000   345 ;       CHECK = Comparison is needed (YES,NO)
0000   346 ;
0000   347         .IF BLANK type
0000   348          .WORD  0
0000   349         .IF_FALSE          ; BLANK type
0000   350          $$$typ = type & prm_typ_m_code          ; Isolate type code
0000   351
0000   352          $$$flg = 0
0000   353          .IIF NOT_BLANK <invalid>, $$$flg = $$$flg!prm_flg_m_invalid
0000   354          .IIF IDN <check><YES>, $$$flg = $$$flg!prm_flg_m_check
0000   355          .IIF IDN <base><CDB>, $$$flg = $$$flg!prm_flg_m_cdb
0000   356
0000   357          .IF BLANK string
0000   358
0000   359                .WORD   $$$typ
0000   360           .IIF NOT_BLANK         <min>, $$$flg = $$$flg!prm_flg_m_min
0000   361           .IIF NOT_BLANK         <max>, $$$flg = $$$flg!prm_flg_m_max
0000   362                .BYTE   $$$flg
0000   363           $$$off = offset & prm_off_m_value     ; Isolate offset only
0000   364           $$$wid = 0                            ; Set null width
0000   365           .IIF IDN <width><B>, $$$wid = <1@prm_off_v_width>
0000   366           .IIF IDN <width><W>, $$$wid = <2@prm_off_v_width>
0000   367           .IIF IDN <width><L>, $$$wid = <3@prm_off_v_width>
0000   368                .WORD   $$$off!$$$wid
0000   369           .IIF NOT_BLANK         <min>, .WORD    min
0000   370           .IIF NOT_BLANK         <max>, .WORD    max
0000   371           line_prm_bufsiz = line_prm_bufsiz + 6
0000   372
0000   373          .IF_FALSE       ; BLANK STRING
0000   374
0000   375                .WORD   $$$typ!prm_typ_m_string ; Indicate a string parameter
0000   376                .BYTE   $$$flg
0000   377            $$$off = offset & prm_off_m_value    ; Isolate offset only
0000   378            $$$wid = <size @ prm_off_v_width> & prm_off_m_width ; Get max allowed
0000   379            $$$siz = <$$$wid @ -prm_off_v_width>
0000   380                .WORD   $$$off!$$$wid
0000   381            line_prm_bufsiz = line_prm_bufsiz + 4 + $$$siz
0000   382
0000   383           .ENDC           ; BLANK STRING
0000   384
0000   385           .IIF NOT_BLANK <invalid>, .WORD    invalid
0000   386
0000   387          .ENDC            ; BLANK TYPE
0000   388 .ENDM   PARAM
0000   389 ;
0000   390 ; ======================
0000   391 ;
0000   392 .MACRO  OFSET   SIZE,OFFSET,BASE
0000   393          .IF IDN <base><LINE>
0000   394          .WORD   cdb_'size'_'offset'
0000   395          .IFF
0000   396          .WORD   ucb$'size'_XQ_'offset'
0000   397          .ENDC
0000   398 .ENDM   OFSET
0000   399 ;
```

```
0000    400 ; ======================
0000    401 ;
0000    402 .MACRO  COUNTER TYPE,WIDTH=16,OFFSET=0,BASE=LINE,BITMAP
0000    403         .IIF NDF 'base'_ctr_size, 'base'_ctr_size = 0
0000    404         .IIF NDF 'base'_ctr_bufsiz, 'base'_ctr_bufsiz = 0
0000    405         .IF IDN <base><LINE>
0000    406         $$$typ = nma$c_ctlin_'type' & nma$m_cnt_typ
0000    407         .IFF
0000    408         $$$typ = nma$c_ctcir_'type' & nma$m_cnt_typ
0000    409         .ENDC
0000    410         $$$wid = 0                          ; Set reserved mask width
0000    411         .IIF IDN <width><8>, $$$wid = <1@nma$v_cnt_wid>
0000    412         .IIF IDN <width><16>, $$$wid = <2@nma$v_cnt_wid>
0000    413         .IIF IDN <width><32>, $$$wid = <3@nma$v_cnt_wid>
0000    414         .IIF EQ $$$wid, .ERROR              ; Invalid bit width value
0000    415         $$$map = 0
0000    416         .IIF IDN <bitmap><MAP>, $$$map = nma$m_cnt_map
0000    417         .WORD   nma$m_cnt_cou!$$$wid!$$$typ!$$$map
0000    418         .IIF IDN <width><8>, OFSET B,'offset,'base
0000    419         .IIF IDN <width><16>, OFSET W,'offset,'base
0000    420         .IIF IDN <width><32>, OFSET L,'offset,'base
0000    421         'base'_ctr_size = 'base'_ctr_size + 1   ; Tally one more entry
0000    422         'base'_ctr_bufsiz = 'base'_ctr_bufsiz + 2 + <width/8>
0000    423         .IIF IDN <bitmap><MAP>, 'base'_ctr_bufsiz = 'base'_ctr_bufsiz + 2
0000    424 .ENDM   COUNTER
0000    425 ;
0000    426 ; ======================
0000    427 ;
0000    428 .MACRO  MOPCTR  WIDTH=16,OFFSET,BITMAP
0000    429         .IIF NDF mop_ctr_size, mop_ctr_size = 0
0000    430         $$$map = 0
0000    431         $$$wid = width/8
0000    432         .IIF IDN <bitmap><MAP>, $$$map = 1@7
0000    433         .IF NOT_BLANK   <offset>
0000    434             .IIF IDN <width><8>, OFSET B,'offset,LINE
0000    435             .IIF IDN <width><16>, OFSET W,'offset,LINE
0000    436             .IIF IDN <width><32>, OFSET L,'offset,LINE
0000    437             .BYTE       $$$map!$$$wid   ; Counter width in bytes + BITMAP FLAG
0000    438             mop_ctr_size = mop_ctr_size + $$$wid
0000    439             .IIF IDN <bitmap><MAP>, mop_ctr_size = mop_ctr_size + 2
0000    440         .IFF ; NOT_BLANK
0000    441             .WORD       0               ; End of table
0000    442         .ENDC ; NOT_BLANK
0000    443 .ENDM   MOPCTR
0000    444 ;
0000    445 ; ======================
0000    446 ;
0000    447 .MACRO  SKIP    BIT,LOC,REG,CONTEXT=W,?L        ; SKIP FIELD
0000    448         BBC     #BIT,LOC,L                  ; Br if field not present
0000    449         TST'CONTEXT     (REG)+              ; Skip next field
0000    450 L:
0000    451 .ENDM   SKIP
0000    452 ;
0000    453 ; ======================
0000    454 ;
0000    455 .MACRO  $DISPATCH,      INDX,VECTOR,TYPE=W,NMODE=S^#,?MN,?MX,?S,?SS,?ZZ
0000    456 SS:
```

```
0000    457
0000    458          .MACRO  $DSP1,$DSP1_1
0000    459                  .IRP    $DSP1_2,$DSP1_1
0000    460                          $DSP2   $DSP1_2
0000    461                  .ENDR
0000    462          .ENDM   $DSP1
0000    463
0000    464          .MACRO  $DSP2,$DSP2_1,$DSP2_2
0000    465                  .=<$DSP2_1-MN>*2 + S
0000    466                  .WORD   $DSP2_2-S
0000    467          .ENDM   $DSP2
0000    468
0000    469
0000    470          .MACRO  $BND1,$BND1_1,$BND1_2,$BND1_3
0000    471                  $BND2   $BND1_1,$BND1_2
0000    472          .ENDM   $BND1
0000    473
0000    474          .MACRO  $BND2,$BND2_1,$BND2_2
0000    475                  .IIF    $BND2_1,$BND2_2-.,      .=$BND2_2
0000    476          .ENDM   $BND2
0000    477
0000    478          .MACRO  $BND    $BND_1,$BND_2
0000    479                  .IRP    $BND_3,<$BND_2>
0000    480                          $BNDT   $BND_1,$BND_3
0000    481                  .ENDR
0000    482          .ENDM   $BND
0000    483
0000    484          .=0
0000    485 ZZ:
0000    486          $BND    GT,<VECTOR>
0000    487 MX:
0000    488          $BND    LT,<VECTOR>
0000    489 MN:
0000    490          .=SS
0000    491
0000    492 CASE'TYPE        INDX,#<MN-ZZ>,NMODE'<MX-MN>
0000    493 S:
0000    494          .REPT   MX-MN+1
0000    495          .WORD   <MX-MN>*2 + 2
0000    496          .ENDR
0000    497
0000    498          .=S
0000    499
0000    500          $DSP1   <<VECTOR>>
0000    501
0000    502          .=<MX-MN>*2 + S + 2
0000    503
0000    504 .ENDM    $DISPATCH
0000    505
```

```
                     0000   507 ;
                     0000   508 ; Overlays of IRP
                     0000   509 ;
                     0000   510          $DEFINI IRP     GLOBAL
                     0000   511
           00000021  0000   512 . = IRP$W_FUNC+1
                     0021   513 $DEF     IRP$B_XQ_FUNC   .BLKB   1       ; QNA driver internal function code
                     0022   514
           00000038  0022   515 . = IRP$L_MEDIA
                     0038   516 $DEF     IRP$W_XQ_RID                    ; RCV/XMT request ID
                     0038   517 $DEF     IRP$B_XQ_SLOT   .BLKB   1       ; RCV/XMT mapping slot number
                     0039   518 $DEF     IRP$B_XQ_RING   .BLKB   1       ; RCV/XMT ring entry number
                     003A   519
           00000046  003A   520 . = IRP$Q_STATION+6
                     0046   521 $DEF     IRP$B_XQ_DATAP  .BLKB   1       ; XMT buffered data path number
                     0047   522
           0000003C  0047   523 . = IRP$L_MEDIA+4
                     003C   524 $DEF     IRP$L_XQ_SYSBUF .BLKL   1       ; XMT system buffer address
                     0040   525
           0000003C  0040   526 . = IRP$L_MEDIA+4
                     003C   527 $DEF     IRP$L_XQ_DATBUF .BLKL   1       ; User RCV data buffer address
                     0040   528
           00000038  0040   529 . = IRP$L_MEDIA
                     0038   530 $DEF     IRP$W_XQ_USERSIZ .BLKW  1       ; User P2 buffer size on sensemode
                     003A   531
           0000003A  003A   532 . = IRP$L_MEDIA+2
                     003A   533 $DEF     IRP$W_XQ_STATUS .BLKW   1       ; Completion status
                     003C   534
           0000003C  003C   535 . = IRP$L_MEDIA+4
                     003C   536 $DEF     IRP$L_XQ_USERBUF .BLKL  1       ; User P1 buffer address on sensemode
                     0040   537
           00000040  0040   538 . = IRP$Q_STATION
                     0040   539 $DEF     IRP$L_XQ_P2BUF  .BLKL   1       ; User P2 buffer address on sensemode
                     0044   540
           00000044  0044   541 . = IRP$Q_STATION+4
                     0044   542 $DEF     IRP$W_XQ_P2SIZ  .BLKW   1       ; P2 return buffer size on sensemode
                     0046   543
           00000040  0046   544 . = IRP$Q_STATION
                     0040   545 $DEF     IRP$W_XQ_CODE   .BLKW   1       ; Bad parameter code on startup request
                     0042   546
           0000003C  0042   547 . = IRP$L_MEDIA+4
                     003C   548 $DEF     IRP$L_XQ_MAP    .BLKL   1       ; Diagnostics buffer mapping info
                     0040   549
           00000040  0040   550 . = IRP$Q_STATION
                     0040   551 $DEF     IRP$L_XQ_DGUNI  .BLKL   1       ; Diagnostics buffer UNIBUS address
                     0044   552 $DEF     IRP$L_XQ_UPADR  .BLKW   1       ; Micro-process internal address
                     0046   553
           00000094  0046   554 . = IRP$L_RBUFH_AD
                     0094   555 $DEF     IRP$L_XQ_SETUP  .BLKL   1       ; Setup transmit buffer
                     0098   556
           0000003A  0098   557 . = IRP$L_MEDIA+2
                     003A   558 $DEF     IRP$W_XQ_PROTYP .BLKW   1       ; Protocol type for user
                     003C   559
           00000090  003C   560 . = IRP$L_LBOFF
                     0090   561 $DEF     IRP$L_XQ_SHR    .BLKL   1       ; Address of SHR structure for user
                     0094   562
           00000060  0094   563 . = IRP$L_FQFL
```

```
0060    564 $DEF    IRP$C_XQ_STD    .BLKL   1       ; End of "standard" IRP
0064    565
0064    566 ;
0064    567 ; Define driver internal function codes stored in IRP$B_XQ_FUNC of IRP.
0064    568 ; NOTE: These are not really used as bit offsets - but as values.
0064    569 ;
0064    570         _VIELD  XQ_FC,0,<-              ; Internal function codes
0064    571                 <INIT>,-                ; Initialize QNA
0064    572                 <XMIT>,-                ; Transmit request
0064    573                 <RECV>,-                ; Receive request
0064    574                 <STOP>,-                ; Stop protocol
0064    575                 <CANCEL>,-              ; Cancel request
0064    576                 <RESTART>,-             ; Restart PROTOCOL
0064    577                 <CHMODE>,-              ; Change the setup mode
0064    578                 >                       ;
0064    579
0064    580         $DEFEND IRP                     ; End of IRP overlays
```

```
                     0000   582 ;
                     0000   583 ; Overlays of CXB
                     0000   584 ;
                     0000   585         $DEFINI CXB     GLOBAL
                     0000   586
           00000020  0000   587 . = CXB$L_SPARE1
                     0020   588 $DEF     CXB$B_XQ_FUNC   .BLKB   1       ; QNA driver internal function code
           00000022  0021   589                         .BLKB   1       ; SPARE
                     0022   590 $DEF     CXB$W_XQ_RID                    ; RCV/XMT request ID
                     0022   591 $DEF     CXB$B_XQ_SLOT   .BLKB   1       ; RCV/XMT mapping slot number
                     0023   592 $DEF     CXB$B_XQ_RING   .BLKB   1       ; RCV/XMT ring entry number
                     0024   593
                     0024   594 ;
                     0024   595 ; The following overlays are for transmits only
                     0024   596 ;
           00000024  0024   597 . = CXB$L_SPARE0
                     0024   598         ;
                     0024   599         ;  NOTE: The following two fields area overlayed. So if the Low
                     0024   600         ;  Bit is set, then the address is that of a UCB, else it's an IRP.
                     0024   601         ;
                     0024   602 $DEF     CXB$L_T_IRP                     ; Associated IRP address
                     0024   603 $DEF     CXB$L_T_UCB     .BLKL   1       ; Associated UCB address
                     0028   604
           0000003A  0028   605 . = CXB$C_HEADER-<XQ_C_HEADER>
                     003A   606 $DEF     CXB$T_T_DATA    .BLKB   XQ_C_HEADER ; Standard Ethernet header
                     0048   607
                     0048   608 ;
                     0048   609 ; The following overlays are for receives only
                     0048   610 ;
           0000000B  0048   611 . = CXB$B_CODE
                     000B   612 $DEF     CXB$B_R_FLAGS   .BLKB   1       ; Receive message flags
                     000C   613
           0000001C  000C   614 . = CXB$L_END_ACTION
                     001C   615 $DEF     CXB$W_R_NCHAIN  .BLKW   1       ; Number of buffers in chain
                     001E   616
           00000014  001E   617 . = CXB$L_IRP
                     0014   618 $DEF     CXB$W_R_STS     .BLKW   1       ; Receive status
                     0016   619
           00000038  0016   620 . = CXB$C_HEADER - <XQ_C_HEADER+XQ_C_CNTSIZ>
                     0038   621 $DEF     CXB$T_R_DATA                    ; Start of receive data
                     0038   622 $DEF     CXB$G_R_DEST    .BLKW   3       ; Destination node address
                     003E   623 $DEF     CXB$G_R_SRC     .BLKW   3       ; Source node address
                     0044   624 $DEF     CXB$W_R_PTYPE   .BLKW   1       ; Protocol Type
                     0046   625 $DEF     CXB$T_R_USERDAT                 ; Start of user data
                     0046   626 $DEF     CXB$W_R_SIZE    .BLKW   1       ; Size of received message (if padded).
                     0048   627
                     0048   628 ;
                     0048   629 ; NOTE: The CXB functions are the same as for an IRP (IRP$B_XQ_FUNC)
                     0048   630 ;
                     0048   631
                     0048   632         $DEFEND CXB                     ; End of CXB overlays
```

```
                     0000   634 ;
                     0000   635 ; Definitions that follow the standard UCB fields
                     0000   636 ;
                     0000   637
                     0000   638         $DEFINI UCB     GLOBAL          ; Start of UCB definitions
                     0000   639
00000098             0000   640 . = UCB$C_NI_LENGTH                     ; Position at end of UCB NI extension
                     0098   641
                     0098   642 $DEF     UCB$Q_XQ_QUEUES                 ; Message and I/O request queue heads
                     0098   643 $DEF     UCB$Q_XQ_SHARE  .BLKQ   1       ; List of shared users
                     00A0   644 $DEF     UCB$Q_XQ_IOQS                   ; Start of the I/O queues
                     00A0   645 $DEF     UCB$Q_XQ_RCVMSG .BLKQ   1       ; Receive messages completed
                     00A8   646 $DEF     UCB$Q_XQ_RCVREQ .BLKQ   1       ; Receive IRP waiting for messages
                     00B0   647 $DEF     UCB$Q_XQ_XMTREQ .BLKQ   1       ;% Xmit IRP wait queue (PT-TO-PT)
00000004             00B8   648 UCB$C_XQ_QUEUES = <.-UCB$Q_XQ_QUEUES>/8 ; Number of queue heads
                     00B8   649
                     00B8   650 $DEF     UCB$L_XQ_PID    .BLKL   1       ; Starter's PID
                     00BC   651 $DEF     UCB$L_XQ_CPID   .BLKL   1       ; Creator's PID
                     00C0   652 $DEF     UCB$L_XQ_AST    .BLKL   1       ; Attention AST list
                     00C4   653 $DEF     UCB$L_XQ_DEFUSR .BLKL   1       ; Default shared user (shared use only)
                     00C8   654 $DEF     UCB$W_XQ_QUOTA  .BLKW   1       ; Receive buffer quota
                     00CA   655 $DEF     UCB$W_XQ_PROTYP .BLKW   1       ; Ethernet protocol type
                     00CC   656
                     00CC   657 $DEF     UCB$B_XQ_SETPRM                 ; Start of parameter section
                     00CC   658 $DEF     UCB$G_XQ_DES    .BLKW   3       ; Destination address for shared user
                     00D2   659 $DEF     UCB$W_XQ_HBQ    .BLKW   1       ; Hardware buffer quota
                     00D4   660 $DEF     UCB$B_XQ_ACC    .BLKB   1       ; Protocol access mode
                     00D5   661 $DEF     UCB$B_XQ_BFN    .BLKB   1       ; Number of receive buffers
                     00D6   662
                     00D6   663 $DEF     UCB$B_XQ_SHRPRM                 ; Start of "shared user" validated prms
                     00D6   664 $DEF     UCB$W_XQ_BSZ    .BLKW   1       ; Device buffer size
                     00D8   665 $DEF     UCB$B_XQ_PRO    .BLKB   1       ; Protocol selection
                     00D9   666 $DEF     UCB$B_XQ_PAD    .BLKB   1       ; Padding mode
                     00DA   667 $DEF     UCB$B_XQ_PRM    .BLKB   1       ; Promiscuous mode
                     00DB   668 $DEF     UCB$B_XQ_MLT    .BLKB   1       ; Multicast (all) address state
                     00DC   669 $DEF     UCB$B_XQ_DCH    .BLKB   1       ; Data chaining on receives
                     00DD   670
                     00DD   671 $DEF     UCB$B_XQ_CDBPRM                 ; Start of settable parameters for CDB
                     00DD   672 $DEF     UCB$B_XQ_CON    .BLKB   1       ; Controller mode
00000001             00DE   673 UCB$C_XQ_CDBPRM = .-UCB$B_XQ_CDBPRM
00000008             00DE   674 UCB$C_XQ_SHRPRM = .-UCB$B_XQ_SHRPRM
                     00DE   675
                     00DE   676 $DEF     UCB$G_XQ_PHA    .BLKW   3       ; User defined physical address
00000007             00E4   677 UCB$C_XQ_SETPRM = .-UCB$B_XQ_CDBPRM
                     00E4   678
                     00E4   679 $DEF     UCB$B_XQ_MST    .BLKB   1       ; Maintenance state
                     00E5   680 $DEF     UCB$B_XQ_MULTI  .BLKB   1       ; Number of entries in MULTI
                     00E6   681 $DEF     UCB$B_XQ_MLTTBL .BLKB   1       ; Number of entries in MLTTBL
                     00E7   682 $DEF     UCB$G_XQ_MULTI  .BLKW   3*MAX_C_MLT ; Multicast address list
                     012F   683 $DEF     UCB$G_XQ_MLTTBL .BLKW   3*MAX_C_MLT ; Multicast generation list
                     0177   684
                     0177   685 $DEF     UCB$W_XQ_CTR                    ; Start of counter section
                     0177   686 $DEF     UCB$W_XQ_MNECTR .BLKW   1       ; Multicast address not enabled
                     0179   687 $DEF     UCB$W_XQ_UBUCTR .BLKW   1       ; No buffer available counter
                     017B   688 $DEF     UCB$L_XQ_SBLCTR .BLKL   1       ; Number of blocks sent
                     017F   689 $DEF     UCB$L_XQ_SBYCTR .BLKL   1       ; Number of bytes sent
                     0183   690 $DEF     UCB$L_XQ_RBLCTR .BLKL   1       ; Number of blocks received
```

```
0187   691 $DEF    UCB$L_XQ_RBYCTR .BLKL   1       ; Number of bytes received
018B   692                                         ; Unused/unneeded fields
018B   693 $DEF    UCB$W_XQ_TOTQUO .BLKW   1       ; Total quota for shared UCB
018D   694 $DEF    UCB$L_XQ_FFI    .BLKL   1       ; Fast interface BLOCK address
0191   695 $DEF    UCB$L_XQ_STIRP  .BLKL   1       ;% Address of PT-TO-PT Startup IRP
0195   696
0195   697 $DEF    UCB$C_XQ_LENGTH                 ; Size of XQDRIVER UCB
0195   698
0195   699 ;
0195   700 ; Define device status bits
0195   701 ;
0195   702         $VIELD  UCB,0,<-                ; XQDRIVER UCB$W_DEVSTS bits
0195   703                 <XQ_INITED,,M>,-        ; Device is initialized
0195   704                 <,15>,-                 ; RESERVED
0195   705                 <XQ_PROTYP,,M>,-        ; Protocol type specified
0195   706                 <XQ_SHARE,,M>,-         ; Shared protocol type
0195   707                 <XQ_RUN,,M>,-           ; Unit is in RUN mode
0195   708                 <XQ_START,,M>,-         ;% Unit is in PT-TO-PT startup state
0195   709                 <XQ_STACK,,M>,-         ;% Unit is in PT-TO-PT stack state
0195   710                 <,7>,-                  ; Reserved
0195   711                 <XQ_INTERLOCK,,M>,-     ; RESTART bit is interlocked
0195   712                 <XQ_RESTART,,M>,-       ; Automatic RESTART on PROTOCOL
0195   713                 >                       ;  requested
0195   714
0195   715         $DEFEND UCB                     ; End of UCB definitions
```

```
0000    717 ;
0000    718 ; Device register offsets and bit definitions
0000    719 ;
0000    720         $DEFINI XQ      GLOBAL          ; Start of port CSR definitions
0000    721
0000    722 $DEF    PHYADD0 .BLKW   1               ; Physical address (R/O) - low byte
0002    723 $DEF    PHYADD1 .BLKW   1               ;   more physical address (R/O)
0004    724 $DEF    PHYADD2                         ;   and still more (R/O)
0004    725 $DEF    RCVLIST .BLKW   1               ; Receive descriptor list (W/O)
0006    726 $DEF    RCVLST1                         ;   high order receive list (W/O)
0006    727 $DEF    PHYADD3 .BLKW   1               ;   more physical address (R/O)
0008    728 $DEF    PHYADD4                         ;   more physical address (R/O)
0008    729 $DEF    XMTLIST .BLKW   1               ; Transmit descriptor list (W/O)
000A    730 $DEF    XMTLST1                         ;   high order receive list (W/O)
000A    731 $DEF    PHYADD5 .BLKW   1               ;   more physical address (R/O)
000C    732 $DEF    VECTOR  .BLKW   1               ; Vector addrss (R/W)
000E    733 $DEF    CSR     .BLKW   1               ; Port CSR
0010    734
0010    735         _VIELD  XQ_CSR,0,<-             ; CSR bit definitions
0010    736                 <RCVENA,,M>,-           ; Receive Enable
0010    737                 <RESET,,M>,-            ; Reset
0010    738                 <NXM,,M>,-              ; Non-existent memory
0010    739                 <RROM,,M>,-             ; Read BOOT/DIAGNOSTICS ROM
0010    740                 <XMTINV,,M>,-           ; Transmit list is invalid
0010    741                 <RCVINV,,M>,-           ; Receive list is invalid
0010    742                 <INTENA,,M>,-           ; Interrupt enable
0010    743                 <RCVINT,,M>,-           ; Receive interrupt
0010    744                 <ILOOP,,M>,-            ; Internal loopback (0=ENABLE,1=DISABLE)
0010    745                 <ELOOP,,M>,-            ; External loopback
0010    746                 <SANITY,,M>,-           ; Sanity timer
0010    747                 <,1>,-                  ; RESERVED
0010    748                 <XCAB,,M>,-             ; Transceiver cable okay
0010    749                 <CAR,,M>,-              ; Carrier sense
0010    750                 <ERR,,M>,-              ; Fatal error flag (software set)
0010    751                 <XMTINT,,M>,-           ; Transmit interrupt
0010    752                 >
0010    753
0010    754         _VIELD  XQ_SOFT,0,<-            ; Software error flag bit definitions
0010    755                 <TIMEOUT,,M>,-          ; Timeout
0010    756                 <POWER,,M>,-            ; Powerfail
0010    757                 >
0010    758
0010    759         $DEFEND XQ                      ; End of device register definitions
```

```
                     0000   761
                     0000   762 ;
                     0000   763 ; Define the Transmit Ring Entry
                     0000   764 ;
                     0000   765          $DEFINI XMT     GLOBAL          ; Start of Transmit Ring Entry format
                     0000   766
                     0000   767 $DEF     XMT_W_FLAG      .BLKW   1       ; Flags word
                     0002   768 $DEF     XMT_W_ADDRHI    .BLKW   1       ; Buffer address (high) and descriptor
                     0004   769 $DEF     XMT_W_ADDR      .BLKW   1       ; Buffer address (low 16 bits)
                     0006   770 $DEF     XMT_W_LEN       .BLKW   1       ; 2's complement WORD size
                     0008   771 $DEF     XMT_W_STS       .BLKW   1       ; XMIT Status word
                     000A   772 $DEF     XMT_W_TDR       .BLKW   1       ; Time Domain Reflectometry word
                     000C   773 $DEF     XMT_C_LENGTH                    ; Size of transmit buffer ring entry
          0000003C   000C   774 XMT_K_LENGTH = XMT_C_LENGTH * <MAX_C_XMT+1> ; Size of xmit ring (1 for chain)
                     000C   775
                     000C   776          _VIELD  XMT_FLG,0,<-            ; Define flag bits
                     000C   777                  <,14>,-                 ; RESERVED
                     000C   778                  <ERR,,M>,-              ; Transmit error
                     000C   779                  <LAST,,M>,-             ; LAST packet/NOT used indicator
                     000C   780                                         -;  The driver only queues one seqment
                     000C   781                                         -;  transmit buffers, so this is
                     000C   782                                         -;  essentially an OWN flag.
                     000C   783                  >
                     000C   784
                     000C   785          _VIELD  XMT_DSC,0,<-            ; Define bits for descriptor word
                     000C   786                  <,6>,-                  ; RESERVED for high order address
                     000C   787                  <BEGODD,1,M>,-          ; Buffer begins on a ODD address
                     000C   788                  <ENDODD,1,M>,-          ; Buffer ends on an ODD address
                     000C   789                  <,4>,-                  ; RESERVED
                     000C   790                  <SETUP,1,M>,-           ; Setup operation
                     000C   791                  <EOM,1,M>,-             ; End of message.
                     000C   792                  <CHAIN,1,M>,-           ; Chain operation
                     000C   793                  <VALID,1,M>,-           ; Valid buffer address
                     000C   794                  >
                     000C   795
                     000C   796          _VIELD  XMT_STS,0,<-            ; Define bits for status word
                     000C   797                  <,4>,-                  ; RESERVED
                     000C   798                  <COL,4,M>,-             ; Number of collisions
                     000C   799                  <FAIL,,>,-              ; Collision check failure
                     000C   800                  <ABORT,,>,-             ; Transmission was aborted due to
                     000C   801                                         -;  excessive collisions
                     000C   802                  <,1>,-                  ; RESERVED
                     000C   803                  <NOCAR,,M>,-            ; No carrier ever present
                     000C   804                  <LCAR,,M>,-             ; Loss of carrier
                     000C   805                  <,1>,-                  ; RESERVED
                     000C   806                  <ERR,,M>,-              ; Error on transmit
                     000C   807                  <LAST,,M>,-             ; LAST packet/NOT used indicator
                     000C   808                                         -;  The driver only queues one seqment
                     000C   809                                         -;  transmit buffers, so this is
                     000C   810                                         -;  essentially an OWN flag.
                     000C   811                  >
                     000C   812
                     000C   813          _VIELD  XMT_TDR,0,<-            ; Define bits for TDR WORD
                     000C   814                  <TDR,14,M>,-            ; Time Domain Reflectometry
                     000C   815                  <,2>,-                  ; RESERVED
                     000C   816                  >
                     000C   817
```

```
                     000C   818          $DEFEND XMT                     ; End of Transmit Ring Entry
                     0000   819 ;
                     0000   820 ; Define the Receive List Entry
                     0000   821 ;
                     0000   822          $DEFINI RCV     GLOBAL          ; Start of Receive List Entry format
                     0000   823
                     0000   824 $DEF      RCV_W_FLAG      .BLKW   1       ; Flags word
                     0002   825 $DEF      RCV_W_ADDRHI    .BLKW   1       ; Buffer address (high) and descriptor
                     0004   826 $DEF      RCV_W_ADDR      .BLKW   1       ; Buffer address (low 16 bits)
                     0006   827 $DEF      RCV_W_LEN       .BLKW   1       ; 2's complement WORD size
                     0008   828 $DEF      RCV_W_STS       .BLKW   1       ; Status word
                     000A   829 $DEF      RCV_W_LENB      .BLKW   1       ; Receive length byte <7:0>
                     000C   830 $DEF      RCV_C_LENGTH                    ; Size of transmit buffer ring entry
                0000006C 000C   831 RCV_K_LENGTH = RCV_C_LENGTH * <MAX_C_RCV+1> ; Size of receive ring (1 for chain)
                     000C   832
                     000C   833          _VIELD  RCV_FLG,0,<-            ; Define flag bits
                     000C   834                  <,14>,-                 ; RESERVED
                     000C   835                  <ERR,,M>,-              ; Receive error
                     000C   836                  <LAST,,M>,-             ; LAST packet/NOT used indicator
                     000C   837                  >
                     000C   838
                     000C   839          _VIELD  RCV_DSC,0,<-            ; Define bits for descriptor word
                     000C   840                  <,14>,-                 ; RESERVED for high order address
                     000C   841                  <CHAIN,1,M>,-           ; Chain operation
                     000C   842                  <VALID,1,M>,-           ; Valid buffer address
                     000C   843                  >
                     000C   844
                     000C   845          _VIELD  RCV_STS,0,<-            ; Define status word
                     000C   846                  <OVF,,M>,-              ; DEQNA receive overflow
                     000C   847                  <CRCERR,,M>,-           ; CRC error
                     000C   848                  <FRAME,,M>,-            ; Framing error
                     000C   849                  <SHORT,,M>,-            ; Short on Ethernet Cable
                     000C   850                  <,4>,-                  ; RESERVED
                     000C   851                  <RLEN,3,M>,-            ; Receive length bits <10:8>
                     000C   852                  <RUNT,,M>,-             ; RUNT packet
                     000C   853                  <DISCARD,,M>,-          ; DISCARD packet (VALIDATES OVF & CRCERR)
                     000C   854                  <ESETUP,,M>,-           ; End of setup
                     000C   855                  <ERR,,M>,-              ; Error/USED indicator
                     000C   856                  <LAST,,M>,-             ; LAST packet/NOT used indicator
                     000C   857                  >
                     000C   858
                     000C   859          $DEFEND RCV                     ; End of Receive Ring Entry
                     0000   860
                     0000   861 ;
                     0000   862 ; Transmit Buffer Header Format
                     0000   863 ;
                     0000   864          $DEFINI XBUF                    ; Define transmit buffer header
                     0000   865
                     0000   866 $DEF      XBUF_G_DEST     .BLKW   3       ; Destination address
                     0006   867 $DEF      XBUF_G_SRC      .BLKW   3       ; Source address (overlays UCB)
                     000C   868 $DEF      XBUF_W_TYPE     .BLKW   1       ; Protocol type
                     000E   869 $DEF      XBUF_T_DATA                     ; Start of xmit data
                     000E   870 $DEF      XBUF_C_HEADER                   ; Size of buffer header
                     000E   871
                     000E   872 $DEF      XBUF_W_SIZE     .BLKW   1       ; Size of buffer (only if padding)
                     0010   873
                     0010   874          $DEFEND XBUF                    ; End of transmit buffer header
```

```
                        ;
                        ; Block header for non-DECnet xmit buffers
                        ;
                                $DEFINI BLK                             ; Define a standard block header
                        
0000    $DEF            BLK_L_LINK      .BLKL   2                       ; Forward and backward queue links
0008    $DEF            BLK_W_SIZE      .BLKW   1                       ; Block size
000A    $DEF            BLK_B_TYPE      .BLKB   1                       ; Block type
000B    $DEF            BLK_B_SPARE     .BLKB   1                       ; SPARE byte
000C    $DEF            BLK_T_DATA                                      ; Start of data
000C    $DEF            BLK_C_HEADER                                    ; Size of buffer header
000C
000C                    $DEFEND BLK
```

```
                     ;
                     ; Define the DEQNA Controller Data Block (CDB) fields
                     ;

                             $DEFINI CDB     GLOBAL          ; Start of CDB definitions

0000   $DEF     CDB_L_FQFL      .BLKL   1       ; Fork queue forward link
0004   $DEF     CDB_L_FQBL      .BLKL   1       ; Fork queue backward link
0008   $DEF     CDB_W_SIZE      .BLKW   1       ; Size of CDB
000A   $DEF     CDB_B_TYPE      .BLKB   1       ; Type of structure
000B   $DEF     CDB_B_FIPL      .BLKB   1       ; Fork IPL
000C   $DEF     CDB_L_FPC       .BLKL   1       ; Fork PC
0010   $DEF     CDB_L_CSR                       ; Port CSR contents
0010   $DEF     CDB_L_FR3       .BLKL   1       ; Fork R3
0014   $DEF     CDB_L_FR4       .BLKL   1       ; Fork R4
0018   $DEF     CDB_B_NEXTXMT   .BLKB   1       ; Next entry in XMT ring
0019   $DEF     CDB_B_NEXTRCV   .BLKB   1       ; Next entry in RCV ring
001A   ASSUME   MAX_C_XMT LE 8
001A   ASSUME   MAX_C_RCV LE 8
001A   $DEF     CDB_B_RCVMAP    .BLKB   1       ; RCV map slot in use flags
001B   $DEF     CDB_B_XMTMAP    .BLKB   1       ; XMT map slot in use flags
001C   $DEF     CDB_L_RCVMAP    .BLKL   MAX_C_RCV-1 ; RCV mapping vector
0038   $DEF     CDB_L_XMTMAP    .BLKL   MAX_C_XMT-1 ; XMT mapping vector
0044   $DEF     CDB_L_RRINGPA   .BLKL   MAX_C_RCV+1 ; RCV Ring entry PHYSICAL address
0068   $DEF     CDB_L_XRINGPA   .BLKL   MAX_C_XMT+1 ; XMT Ring entry PHYSICAL address
007C   $DEF     CDB_L_RRINGVA   .BLKL   MAX_C_RCV ; RCV Ring entry VIRTUAL address
009C   $DEF     CDB_L_XRINGVA   .BLKL   MAX_C_XMT ; XMT Ring entry VIRTUAL address
00AC   $DEF     CDB_L_RCV_PA    .BLKL   MAX_C_RCVUV1 ; Receive contiguous buffer
00C0                                            ;  physical address
00C0   $DEF     CDB_L_XMT_PA    .BLKL   MAX_C_XMTUV1 ; Transmit contiguous buffer
00C4                                            ;  physical address
00C4   $DEF     CDB_L_RCV_VA    .BLKL   MAX_C_RCVUV1 ; Receive contiguous buffer
00D8                                            ;  virtual address
00D8   $DEF     CDB_L_XMT_VA    .BLKL   MAX_C_XMTUV1 ; Transmit contiguous buffer
00DC                                            ;  virtual address
00DC   $DEF     CDB_Q_QUEUES                    ; Start of CDB queues
00DC   $DEF     CDB_Q_XMTREQ    .BLKQ   1       ; Transmit request queue
00000001 00E4   CDB_C_ABORTS = <.-CDB_Q_QUEUES>/8       ; Number of Queues to abort requests
00E4   $DEF     CDB_Q_INPUT     .BLKQ   1       ; Input process queue
00EC   $DEF     CDB_Q_XMTPND    .BLKQ   1       ; Transmit pending queue
00F4   $DEF     CDB_Q_RCVBUF    .BLKQ   1       ; Receive buffer queue
00FC   $DEF     CDB_Q_RCVPND    .BLKQ   1       ; Receive pending queue
0104   $DEF     CDB_Q_POST      .BLKQ   1       ; Post process queue
00000006 010C   CDB_C_QUEUES = <.-CDB_Q_QUEUES>/8       ; Number of Queue Heads
010C
010C   $DEF     CDB_B_LASTRCV   .BLKB   1       ; Last entry done in RCV ring
010D   $DEF     CDB_B_LASTXMT   .BLKB   1       ; Last entry done in XMT ring
010E   $DEF     CDB_B_RCVCNT    .BLKB   1       ; Count of receives given to QNA
010F   $DEF     CDB_B_XMTCNT    .BLKB   1       ; Count of xmits given to QNA
0110   $DEF     CDB_W_BSZ       .BLKW   1       ; Device buffer size
0112   $DEF     CDB_W_QUOTA     .BLKW   1       ; SYSTEM buffer quota
0114
0114   $DEF     CDB_L_DEVDEPEND .BLKL   1       ; Device dependent longword
0118   $DEF     CDB_L_UCB0      .BLKL   1       ; Address of UCB #0
011C   $DEF     CDB_B_SPARE     .BLKB   1       ; SPARE BYTE
011D   $DEF     CDB_B_DIAG1     .BLKB   1       ; Diagnostic info byte
011E   $DEF     CDB_W_DIAG2     .BLKW   1       ; Second word of diagnostic info
```

```
                     0120   947
                     0120   948 $DEF    CDB_G_COUNTER                       ; Start of counters
                     0120   949 $DEF    CDB_W_ZERO      .BLKW   1           ; Seconds since last zeroed
                     0122   950 $DEF    CDB_L_DBRCTR    .BLKL   1           ; Messages received
                     0126   951 $DEF    CDB_L_MBLCTR    .BLKL   1           ; Multicast messages received
                     012A   952 $DEF    CDB_W_RFLMAP    .BLKW   1           ; Messages received in error - bitmap
                     012C   953 $DEF    CDB_W_RFLCTR    .BLKW   1           ; Messages received in error
                     012E   954         ASSUME  CDB_W_RFLCTR EQ CDB_W_RFLMAP+2
                     012E   955 $DEF    CDB_L_BRCCTR    .BLKL   1           ; Bytes received
                     0132   956 $DEF    CDB_L_MBYCTR    .BLKL   1           ; Multicast bytes received
                     0136   957 $DEF    CDB_W_OVRCTR    .BLKW   1           ; Packets lost due to hardware buffers
                     0138   958 $DEF    CDB_W_LBECTR    .BLKW   1           ; Packets lost due to system buffer error
                     013A   959 $DEF    CDB_L_DBSCTR    .BLKL   1           ; Messages sent
                     013E   960 $DEF    CDB_L_MBSCTR    .BLKL   1           ; Multicast messages sent
                     0142   961 $DEF    CDB_L_BSMCTR    .BLKL   1           ; Messages sent - several errors
                     0146   962 $DEF    CDB_L_BS1CTR    .BLKL   1           ; Messages sent - 1 error
                     014A   963 $DEF    CDB_L_BIDCTR    .BLKL   1           ; Messages sent - initially deferred
                     014E   964 $DEF    CDB_L_BSNCTR    .BLKL   1           ; Bytes sent
                     0152   965 $DEF    CDB_L_MSNCTR    .BLKL   1           ; Multicast bytes sent
                     0156   966 $DEF    CDB_W_SFLMAP    .BLKW   1           ; Send failures - bitmap
                     0158   967 $DEF    CDB_W_SFLCTR    .BLKW   1           ; Send failures
                     015A   968         ASSUME  CDB_W_SFLCTR EQ CDB_W_SFLMAP+2
                     015A   969 $DEF    CDB_W_CDCCTR    .BLKW   1           ; Transmit collision check failure
                     015C   970
                     015C   971 $DEF    CDB_W_UFDCTR    .BLKW   1           ; No protocol type counter on receive
                     015E   972 $DEF    CDB_W_SBUCTR    .BLKW   1           ; System buffer unavailable
                     0160   973 $DEF    CDB_W_UBUCTR    .BLKW   1           ; No buffer available on all PTs
                     0162   974
                     0162   975 $DEF    CDB_L_RINGMAP   .BLKL   1           ; Mapping information for RINGs
                     0166   976 ;
                     0166   977 ; Define the UNIBUS mapped portion of the CDB (QNA data structures)
                     0166   978 ;
                     0166   979 $DEF    CDB_G_MAPPED                        ; Start of UNIBUS mapped portion of CDB
                     0166   980 $DEF    CDB_G_RRING     .BLKB   RCV_K_LENGTH    ; Recv ring buffer
                     01D2   981 $DEF    CDB_G_XRING     .BLKB   XMT_K_LENGTH    ; Xmit ring buffer
000000A8             020E   982 CDB_C_MAPPED = .-CDB_G_MAPPED                 ; Size of UNIBUS mapped portion of CDB
                     020E   983 $DEF    CDB_C_ZERO                          ; Size of CDB to zero (everything from
                     020E   984                                             ;  the beginning to this point).
                     020E   985 $DEF    CDB_B_TIM_XMT   .BLKB   1           ; Transmit timer cell
                     020F   986 $DEF    CDB_B_UNTCNT    .BLKB   1           ; Number of active units (UCBs)
                     0210   987 $DEF    CDB_L_UV1BUF    .BLKL   1           ; u-VAX I buffer area address
                     0214   988 $DEF    CDB_L_PRMUSER   .BLKL   1           ; Promiscuous user's UCB address
                     0218   989 $DEF    CDB_L_TQE       .BLKB   TQE$C_LENGTH ; Timer queue element
                     0248   990 $DEF    CDB_W_MODE      .BLKW   1           ; QNA hardware mode
                     024A   991 $DEF    CDB_B_STS       .BLKB   1           ; Controller status
                     024B   992 $DEF    CDB_B_PRM       .BLKB   1           ; Promiscuous mode
                     024C   993 $DEF    CDB_B_MLT       .BLKB   1           ; Multicast (all) address state
                     024D   994
                     024D   995 $DEF    CDB_B_SETPRM                        ; Start of settable parameters
                     024D   996 $DEF    CDB_B_CON       .BLKB   1           ; Controller mode
00000001             024E   997 CDB_C_SETPRM = .-CDB_B_SETPRM                ; Size of settable parameter list
                     024E   998
                     024E   999 $DEF    CDB_G_PHA       .BLKW   3           ; User defined physical address
                     0254  1000 $DEF    CDB_G_HWA       .BLKW   3           ; Hardware physical address
                     025A  1001 $DEF    CDB_G_PHYADR    .BLKW   3           ; The current hardware address
                     0260  1002 $DEF    CDB_B_MULTI     .BLKB   1           ; Number of entries in Multicast list
                     0261  1003 $DEF    CDB_B_MLTTBL    .BLKB   1           ; Number of entries in MLTTBL
```

```
0262  1004 $DEF     CDB_G_MULTI     .BLKW   3*MAX_C_MLT ; Multicast address table
02AA  1005 $DEF     CDB_G_MLTTBL    .BLKW   3*MAX_C_MLT ; Multicast address generation table
02F2  1006 $DEF     CDB_B_AQUOTA    .BLKB   1           ; Additional QUOTA available
02F3  1007 $DEF     CDB_B_MQUOTA    .BLKB   1           ; Maximum extra QUOTA allowed
02F4  1008
02F4  1009 $DEF     CDB_C_LENGTH                        ; Size of QNA CDB
02F4  1010
02F4  1011          _VIELD  CDB_STS,0,<                 ; CDB status bit for CDB_B_STS
02F4  1012                  <INITED,,M>,-               ; Inited
02F4  1013                  <RUN,,M>,-                  ; Device is running
02F4  1014                  <FORK_PEND,,M>,-            ; Fork process is pending
02F4  1015                  <TIMER,,M>,-                ; Timer is active
02F4  1016                  <ERR,,M>,-                  ; Fatal error has occurred
02F4  1017                  <SETUP,,M>,-                ; Device is in SETUP mode
02F4  1018                  >
02F4  1019
02F4  1020          _VIELD  CDB_MOD,0,<                 ; Bits for mode definition CDB_W_MODE
02F4  1021                  <MULTI,,M>,-                ; All multicasts enabled
02F4  1022                  <PROM,,M>,-                 ; Promiscuous mode enabled
02F4  1023                  >
02F4  1024
02F4  1025          $DEFEND CDB                         ; End of QNA CDB definitions
```

```
                     ;
                     ; P2 buffer header definition
                     ;
                             $DEFINI P2B
                     
0000  1032 $DEF      P2B_L_POINTER   .BLKL   1       ; Pointer to start of data
0004  1033 $DEF      P2B_L_BUFFER    .BLKL   1       ; Address of user's data buffer
0008  1034 $DEF      P2B_W_SIZE      .BLKW   1       ; Size of P2 buffer
000A  1035 $DEF      P2B_B_TYPE      .BLKB   1       ; Type of structure
000B  1036 $DEF      P2B_B_SPARE     .BLKB   1       ; Spare byte
000C  1037 $DEF      P2B_C_LENGTH                    ; Size of P2 buffer header
000C  1038 $DEF      P2B_T_DATA                      ; Start of data
000C  1039
000C  1040           $DEFEND P2B
                     ;
                     ; Diagnostics buffer definition
                     ;
                             $DEFINI DIAG
                     ;
                     ; Driver independent portion of diagnostics buffer
                     ;
0000  1048 $DEF      DIAG_L_DATA     .BLKL   1       ; Pointer to start of data
0004  1049 $DEF      DIAG_L_BUFFER   .BLKL   1       ; User buffer address
0008  1050 $DEF      DIAG_W_SIZE     .BLKW   1       ; Size of structure
000A  1051 $DEF      DIAG_B_TYPE     .BLKB   1       ; Type of structure
000B  1052 $DEF      DIAG_B_SPARE    .BLKB   1       ; Spare byte
000C  1053 $DEF      DIAG_T_DATA                     ; Start of data
000C  1054 $DEF      DIAG_Q_START    .BLKQ   1       ; Start time for QIO
0014  1055 $DEF      DIAG_Q_FINISH   .BLKQ   1       ; Finish time for QIO
001C  1056 $DEF      DIAG_L_ERRS     .BLKL   1       ; Number of device errors
0020  1057 $DEF      DIAG_L_EXTRA    .BLKL   1       ; Number of longwords that follow
0024  1058 ;
0024  1059 ; Driver dependent portion of diagnostics buffer
0024  1060 ;
0024  1061 $DEF      DIAG_L_DEPEND
0024  1062 $DEF      DIAG_W_CSR      .BLKW   1       ; Last port CSR contents
0026  1063 $DEF      DIAG_W_ERR      .BLKW   1       ; Ring entry error summary
0028  1064 $DEF      DIAG_W_ERR2     .BLKW   1       ; Extra ring entry error info
002A  1065 $DEF      DIAG_G_HWA      .BLKW   3       ; Hardware physical address
0030  1066 ;
0030  1067 ; The following is valid only on read (receive) QIOs
0030  1068 ;
0030  1069 $DEF      DIAG_T_RDATA                    ; Start of receive data
0030  1070 $DEF      DIAG_G_DEST     .BLKW   3       ; Destination address
0036  1071 $DEF      DIAG_G_SRC      .BLKW   3       ; Source address
003C  1072 $DEF      DIAG_W_TYPE     .BLKW   1       ; Protocol type
003E  1073 $DEF      DIAG_C_LENGTH                   ; Start of data
00000006  003E  1074 DIAG_C_EXTRA = .-DIAG_L_DEPEND/4
003E  1075
003E  1076           $DEFEND DIAG
                     ;
                     ; Receive buffer header definition
                     ;
                             $DEFINI RHDR
                     
0000  1082 $DEF      RHDR_L_DATA     .BLKL   1       ; Pointer to start of data
0004  1083 $DEF      RHDR_L_BUFFER   .BLKL   1       ; User buffer address
```

```
                     0008  1084 $DEF     RHDR_W_SIZE     .BLKW   1       ; Size of structure
                     000A  1085 $DEF     RHDR_B_TYPE     .BLKB   1       ; Type of structure
                     000B  1086 $DEF     RHDR_B_SPARE    .BLKB   1       ; Spare byte
                     000C  1087 $DEF     RHDR_T_DATA                     ; Start of data
                     000C  1088 $DEF     RHDR_G_DEST     .BLKW   3       ; Destination address
                     0012  1089 $DEF     RHDR_G_SRC      .BLKW   3       ; Source address
                     0018  1090 $DEF     RHDR_W_TYPE     .BLKW   1       ; Protocol type
                     001A  1091 $DEF     RHDR_C_LENGTH                   ; Start of data
            0000000E 001A  1092 RHDR_C_DATA = .-RHDR_T_DATA
                     001A  1093
                     001A  1094         $DEFEND RHDR
                     0000  1095
                     0000  1096 ;
                     0000  1097 ; Shareable protocol type data structure
                     0000  1098 ;
                     0000  1099         $DEFINI SHR     GLOBAL
                     0000  1100 $DEF     SHR_L_QFL       .BLKL   1       ; Forward link pointer
                     0004  1101 $DEF     SHR_L_QBL       .BLKL   1       ; Backward link pointer
                     0008  1102 $DEF     SHR_W_SIZE      .BLKW   1       ; Size of structure
                     000A  1103 $DEF     SHR_B_TYPE      .BLKB   1       ; Type of structure
                     000B  1104 $DEF     SHR_B_STS       .BLKB   1       ; SHR status
                     000C  1105 $DEF     SHR_L_PID       .BLKL   1       ; User's PID
                     0010  1106 $DEF     SHR_W_CHAN      .BLKW   1       ; User's channel
                     0012  1107 $DEF     SHR_G_DEST      .BLKW   3       ; Destination address
                     0018  1108 $DEF     SHR_Q_QUEUES                    ; Start of queues
                     0018  1109 $DEF     SHR_Q_RCVMSG    .BLKQ   1       ; Received messages waiting for IRPs
                     0020  1110 $DEF     SHR_Q_RCVREQ    .BLKQ   1       ; Receive IRPs waiting for messages
            00000002 0028  1111 SHR_C_QUEUES = <.-SHR_Q_QUEUES>/8       ; Number of queues
                     0028  1112 $DEF     SHR_W_QUOTA     .BLKW   1       ; User's shared quota
                     002A  1113 $DEF     SHR_C_LENGTH                    ; Size of data structure
                     002A  1114
                     002A  1115         _VIELD  SHR_STS,0,<-            ; SHR status bits
                     002A  1116                 <INITED,,M>,-           ; Protocol type is inited
                     002A  1117                 >
                     002A  1118
                     002A  1119         $DEFEND SHR
```

```
                        .SBTTL  Standard tables
                ;
                ; Driver prologue table
                ;
                        DPTAB   -                               ;
                                END=XQ_END,-                    ; END OF DRIVER
                                ADAPTER=UBA,-                   ; ADAPTER TYPE
                                UCBSIZE=UCB$C_XQ_LENGTH,-       ; SIZE OF UCB
                                NAME=XQDRIVER                   ; DRIVER NAME
                ;
                        DPT_STORE INIT                          ; START OF CONTROLLER INIT
                        DPT_STORE UCB,UCB$B_FIPL,B,IPL$_XQ_FIPL ; FORK IPL
                        DPT_STORE UCB,UCB$B_DIPL,B,IPL$_XQ_DIPL ; DEVICE IPL
                        DPT_STORE ORB,ORB$B_FLAGS,B,-           ; Protection block flags
                                  <ORB$M_PROT_16>               ; SOGW protection word
                        DPT_STORE ORB,ORB$W_PROT,W,0            ; default protection
                        DPT_STORE ORB,ORB$L_OWNER,L,0           ; no owner as yet
                        DPT_STORE UCB,UCB$L_DEVCHAR,L,- ; DEVICE CHARACTERISTICS
                                 <DEV$M_SHR!-
                                  DEV$M_NET!-
                                  DEV$M_AVL!-
                                  DEV$M_IDV!-
                                  DEV$M_ODV>
                ;
                        DPT_STORE UCB,UCB$B_DEVCLASS,B,DC$_SCOM ; Device class
                        DPT_STORE UCB,UCB$B_DEVTYPE,B,DT$_DEQNA ; Device type
                        DPT_STORE UCB,UCB$W_DEVBUFSIZ,W,512     ; Default buffer size
                        DPT_STORE UCB,UCB$W_STS,W,<UCB$M_ONLINE!UCB$M_TEMPLATE>
                        DPT_STORE UCB,UCB$G_XQ_PHA,L,-1         ; No default physical address
                        DPT_STORE UCB,UCB$G_XQ_PHA+4,W,-1       ;        ...
                ;
                ; Store defaults for all parameters
                ;
                        DPT_STORE UCB,UCB$W_XQ_BSZ,W,1500       ; Default device buffer size
                        DPT_STORE UCB,UCB$B_XQ_BFN,B,1          ; Default user buffer number
                        DPT_STORE UCB,UCB$W_XQ_HBQ,W,INIT_C_QUOTA ; Hardware Buffer Quota
                        DPT_STORE UCB,UCB$B_XQ_PRO,B,NMA$C_LINPR_NI ; "NI" is the protocol mode
                        DPT_STORE UCB,UCB$B_XQ_PRM,B,NMA$C_STATE_OFF ; Promiscuous mode is OFF
                        DPT_STORE UCB,UCB$B_XQ_MLT,B,NMA$C_STATE_OFF ; All multicasts is OFF
                        DPT_STORE UCB,UCB$B_XQ_DCH,B,NMA$C_STATE_ON ; Data chaining is ON
                        DPT_STORE UCB,UCB$B_XQ_PAD,B,NMA$C_STATE_ON ; Padding is ON
                        DPT_STORE UCB,UCB$B_XQ_CON,B,NMA$C_LINCN_NOR ; Controler mode is NORMAL
                        DPT_STORE UCB,UCB$B_XQ_ACC,B,NMA$C_ACC_EXC ; Exclusive mode is default
                ;
                        DPT_STORE REINIT
                ;
                        DPT_STORE DDB,DDB$L_DDT,D,XQ$DDT        ; DDT ADDRESS
                        DPT_STORE CRB,CRB$L_INTD+4,D,QNA_INTR   ; QNA interrupt service routine
                        DPT_STORE CRB,CRB$L_INTD+VEC$L_INITIAL,D,CONTROL_INIT; CONTROLLER INIT ADDRE
                        DPT_STORE CRB,CRB$L_INTD+VEC$L_UNITINIT,D,UNIT_INIT; UNIT INIT
                        DPT_STORE CRB,CRB$L_INTD+VEC$L_START,D,FFI_INIT ; FFI INIT
                        DPT_STORE END
                ;
00000000        .PSECT  $$$115_DRIVER,LONG
                ;
                ; LOCAL STORAGE
                ;
```

```
                        ; Driver dispatch table
                        ;
                                DDTAB   DEVNAM=XQ,-                     ; DRIVER DISPATCH TABLE
                                        START=STARTIO,-                 ; Start I/O operation
                                        FUNCTB=XQ_FUNCTABLE,-           ; Function decision table address
                                        CANCEL=CANCEL,-                 ; CANCEL I/O entry point
                                        REGDMP=REG_DUMP,-               ; Register dump entry point
                                        DIAGBF=<DIAG_C_LENGTH>,-        ; Diagnostic buffer size
                                        CLONEDUCB=CLONED_UCB,-          ; Cloned UCB initialization
                                        ALTSTART=ALT_START              ; Alternate start I/O entry point
                        ;
                        ; Function decision table
                        ;
                        XQ_FUNCTABLE:                                   ;
                                FUNCTAB ,-                              ; Legal Functions
                                <WRITEVBLK,WRITELBLK,WRITEPBLK,READVBLK,READLBLK,-
                                READPBLK,SENSEMODE,SENSECHAR,SETMODE,SETCHAR>
                                FUNCTAB ,-                              ; Buffered Functions
                                <WRITEVBLK,WRITELBLK,WRITEPBLK,READVBLK,READLBLK,-
                                READPBLK,SENSEMODE,SENSECHAR,SETMODE,SETCHAR>
                                FUNCTAB XMT_FDT,<WRITELBLK,WRITEPBLK,WRITEVBLK> ;
                                FUNCTAB RCV_FDT,<READLBLK,READPBLK,READVBLK> ;
                                FUNCTAB SETMODE_FDT,<SETMODE,SETCHAR> ;
                                FUNCTAB SENSEMODE_FDT,<SENSEMODE,SENSECHAR> ;
```

```
                     0078  1204          .SBTTL  Local driver storage
                     0078  1205 ;
                     0078  1206 ; P2 Buffer verification tables
                     0078  1207 ;
                     0078  1208
                     0078  1209          $DEFINI PARAM
                     0000  1210
                     0000  1211 $DEF     PRM_W_TYPE      .BLKW   1       ; Parameter type
                     0002  1212
                     0002  1213          _VIELD  PRM_TYP,0,<-            ; Parameter type field
                     0002  1214                  <CODE,12,M>,-           ; Parameter type code
                     0002  1215                  <STRING,1,M>,-          ; Parameter is a string
                     0002  1216                  >
                     0002  1217
                     0002  1218 $DEF     PRM_B_FLAG      .BLKB   1       ; Parameter flags
                     0003  1219
                     0003  1220          _VIELD  PRM_FLG,0,<-            ; Parameter flag bits
                     0003  1221                  <MIN,1,M>,-             ; Parameter minimum value present
                     0003  1222                  <MAX,1,M>,-             ; Parameter maximum value present
                     0003  1223                  <INVALID,1,M>,-         ; Parameter invalid value is present
                     0003  1224                  <CDB,1,M>,-             ; Offset is in CDB data base
                     0003  1225                  <CHECK,1,M>,-           ; Check values with current
                     0003  1226                  >
                     0003  1227
                     0003  1228 $DEF     PRM_W_OFF       .BLKW   1       ; Parameter offset in structure
                     0005  1229
                     0005  1230          _VIELD  PRM_OFF,0,<-            ; Offset word fields
                     0005  1231                  <VALUE,10,M>,-          ; Offset value
                     0005  1232                  <WIDTH,6,M>,-           ; Size of field in structure
                     0005  1233                  >
                     0005  1234
                     0005  1235          $DEFEND PARAM
                     0078  1236
                     0078  1237 ;
                     0078  1238 ; Define Line parameters
                     0078  1239 ;
00000000             0078  1240 LINE_PRM_BUFSIZ=0                        ; Line parameter buffer size
                     0078  1241 LINE_PARAM_WO:                           ; "Write-Only" line parameters
                     0078  1242
                     0078  1243          PARAM   NMA$C_PCLI_HBQ,-        ; Hardware Buffer Quota
                     0078  1244                  OFFSET=UCB$W_XQ_HBQ,-
                     0078  1245                  WIDTH=W,MAX=16384,-
                     0078  1246                  INVALID=UCB$M_XQ_INITED
                     0081  1247
                     0081  1248 LINE_PARAM:                            ; Start of line parameters
                     0081  1249
                     0081  1250          PARAM   NMA$C_PCLI_ACC,-        ; Access mode for protocol type
                     0081  1251                  OFFSET=UCB$B_XQ_ACC,-
                     0081  1252                  WIDTH=B,-
                     0081  1253                  MIN=NMA$C_ACC_SHR,-
                     0081  1254                  MAX=NMA$C_ACC_EXC
                     008A  1255
                     008A  1256          PARAM   NMA$C_PCLI_PRO,-        ; Protocol selection mode
                     008A  1257                  OFFSET=UCB$B_XQ_PRO,-
                     008A  1258                  WIDTH=B,-
                     008A  1259                  MIN=NMA$C_LINPR_POI,-   ;% Accept either point or NI
                     008A  1260                  MAX=NMA$C_LINPR_NI
```

```
                        1261
                        1262    PARAM   NMA$C_PCLI_BUS,-        ; Buffer size
                        1263            OFFSET=UCB$W_DEVBUFSIZ,-
                        1264            WIDTH=W,-
                        1265            MIN=MIN_PKT_SIZE,-      ; User buffer LIMITS
                        1266            MAX=MAX_PKT_SIZE,-
                        1267            INVALID=UCB$M_XQ_INITED
                        1268
                        1269    PARAM   NMA$C_PCLI_BFN,-        ; Buffer number
                        1270            OFFSET=UCB$B_XQ_BFN,-
                        1271            WIDTH=B,-
                        1272            MIN=0,MAX=255,-
                        1273            INVALID=UCB$M_XQ_INITED
                        1274
                        1275    PARAM   NMA$C_PCLI_PHA,-        ; Physical NI address
                        1276            OFFSET=UCB$G_XQ_PHA,-
                        1277            STRING=YES,-
                        1278            SIZE=<2+6>,-
                        1279            INVALID=UCB$M_XQ_INITED
                        1280
                        1281    PARAM   NMA$C_PCLI_DCH,-        ; Data chaining on receives
                        1282            OFFSET=UCB$B_XQ_DCH,-
                        1283            WIDTH=B,-
                        1284            MAX=NMA$C_STATE_OFF
                        1285
                        1286    PARAM   NMA$C_PCLI_PAD,-        ; Padding mode
                        1287            OFFSET=UCB$B_XQ_PAD,-
                        1288            WIDTH=B,-
                        1289            MAX=NMA$C_STATE_OFF
                        1290
                        1291    PARAM   NMA$C_PCLI_PRM,-        ; Promiscuous mode state
                        1292            OFFSET=UCB$B_XQ_PRM,-
                        1293            WIDTH=B,-
                        1294            MAX=NMA$C_STATE_OFF,-
                        1295            INVALID=UCB$M_XQ_INITED
                        1296
                        1297    PARAM   NMA$C_PCLI_MLT,-        ; Accept all multicast addresses
                        1298            OFFSET=UCB$B_XQ_MLT,-
                        1299            WIDTH=B,-
                        1300            MAX=NMA$C_STATE_OFF,-
                        1301            INVALID=UCB$M_XQ_INITED
                        1302
                        1303    PARAM   NMA$C_PCLI_CON,-        ; Controller mode
                        1304            OFFSET=UCB$B_XQ_CON,-
                        1305            WIDTH=B,-
                        1306            MAX=NMA$C_LINCN_LOO,-
                        1307            INVALID=UCB$M_XQ_INITED
                        1308
                        1309    PARAM   NMA$C_PCLI_PTY,-        ; Protocol type
                        1310            OFFSET=UCB$W_XQ_PROTYP,-
                        1311            WIDTH=W,-
                        1312            INVALID=UCB$M_XQ_INITED
                        1313
                        1314    PARAM   NMA$C_PCLI_MCA,-        ; Multicast address list
                        1315            OFFSET=UCB$G_XQ_MULTI,-
                        1316            STRING=YES,-
                        1317            SIZE=2+<6*MAX_C_MLT>    ; Maximum size of list
```

```
00E5  1318
00E5  1319          PARAM   NMA$C_PCLI_BSZ,-         ; Device buffer size
00E5  1320                  OFFSET=UCB$W_XQ_BSZ,-
00E5  1321                  WIDTH=W,-
00E5  1322                  MIN=MIN_PKT_SIZE,-
00E5  1323                  MAX=MAX_PKT_SIZE,-
00E5  1324                  INVALID=UCB$M_XQ_INITED
00F0  1325
00F0  1326          PARAM   NMA$C_PCLI_DES,-         ; Destination Address for shared
00F0  1327                  OFFSET=UCB$G_XQ_DES,-    ;  Protocol Type
00F0  1328                  STRING=YES,-
00F0  1329                  SIZE=<2+6>
00F5  1330
00F5  1331 ;****
00F5  1332 ; THE FOLLOWING CAN BE ELIMINATED
00F5  1333 ;****
00F5  1334          PARAM   NMA$C_PCLI_CRC,-         ; CRC enabled
00F5  1335                  OFFSET=UCB$B_XQ_MST,-    ;  garbage
00F5  1336                  WIDTH=B,-                ;
00F5  1337                  MAX=NMA$C_STATE_OFF
00FC  1338
00FC  1339          PARAM                            ; End of table
00FE  1340
00FE  1341 CIRCUIT_PARAM:                            ; Start of circuit parameter table
00FE  1342
00FE  1343          PARAM   NMA$C_PCCI_MST,-         ; Maintenance state
00FE  1344                  OFFSET=UCB$B_XQ_MST,-
00FE  1345                  WIDTH=B,-
00FE  1346                  MIN=NMA$C_STATE_ON,-
00FE  1347                  MAX=NMA$C_STATE_OFF
0107  1348
0107  1349          PARAM                            ; End of table
0109  1350
0109  1351 ;
0109  1352 ; Line/circuit counters
0109  1353 ;
0109  1354 LINE_CTR:                                 ; Start of LINE counters
0109  1355          COUNTER         ZER, 16, ZERO    ; Seconds since last zeroed
010D  1356          COUNTER         DBR, 32, DBRCTR  ; Packets received
0111  1357          COUNTER         MBL, 32, MBLCTR  ; Multicast packets received
0115  1358          COUNTER         RFL, 16, RFLMAP,, MAP ; Packets received in error
0119  1359          COUNTER         BRC, 32, BRCCTR  ; Bytes received
011D  1360          COUNTER         MBY, 32, MBYCTR  ; Multicast bytes received
0121  1361          COUNTER         OVR, 16, OVRCTR  ; Receives lost - Internal buffer error
0125  1362          COUNTER         LBE, 16, LBECTR  ; Receives lost - Local buffer error
0129  1363          COUNTER         DBS, 32, DBSCTR  ; Packets transmitted
012D  1364          COUNTER         MBS, 32, MBSCTR  ; Multicast packets transmitted
0131  1365          COUNTER         BSM, 32, BSMCTR  ; Packets transmitted - several errors
0135  1366          COUNTER         BS1, 32, BS1CTR  ; Packets transmitted - 1 error
0139  1367          COUNTER         BID, 32, BIDCTR  ; Packets transmitted - deferred
013D  1368          COUNTER         BSN, 32, BSNCTR  ; Bytes transmitted
0141  1369          COUNTER         MSN, 32, MSNCTR  ; Multicast bytes transmitted
0145  1370          COUNTER         SFL, 16, SFLMAP,, MAP ; Transmit packets aborted
0149  1371          COUNTER         CDC, 16, CDCCTR  ; Transmit collision check failure
014D  1372          COUNTER         UFD, 16, UFDCTR  ; Unrecognized frame destination
0151  1373          COUNTER         SBU, 16, SBUCTR  ; System buffer unavailable
0155  1374          COUNTER         UBU, 16, UBUCTR  ; User buffer unavailable
```

```
                     0159  1375
                     0159  1376 CIRC_CTR:                                ; Start of CIRCUIT counters
                     0159  1377         COUNTER         DBS, 32, SBLCTR, CIRC ; Blocks sent
                     015D  1378         COUNTER         BSN, 32, SBYCTR, CIRC ; Bytes sent
                     0161  1379         COUNTER         DBR, 32, RBLCTR, CIRC ; Blocks received
                     0165  1380         COUNTER         BRC, 32, RBYCTR, CIRC ; Bytes received
                     0169  1381         COUNTER         MNE, 16, MNECTR, CIRC ; Multicast address not enabled
                     016D  1382         COUNTER         UBU, 16, UBUCTR, CIRC ; User buffer unavailable
                     0171  1383
                     0171  1384 ;
                     0171  1385 ; MOP read counters return table (in order of COUNTERs returned)
                     0171  1386 ;
                     0171  1387 MOPCTRTAB:                                ; Start of MOP counters
                     0171  1388         MOPCTR          16, ZERO          ; Seconds since last zeroed
                     0174  1389         MOPCTR          32, BRCCTR        ; Bytes received
                     0177  1390         MOPCTR          32, BSNCTR        ; Bytes transmitted
                     017A  1391         MOPCTR          32, DBRCTR        ; Packets received
                     017D  1392         MOPCTR          32, DBSCTR        ; Packets transmitted
                     0180  1393         MOPCTR          32, MBYCTR        ; Multicast bytes received
                     0183  1394         MOPCTR          32, MBLCTR        ; Multicast packets received
                     0186  1395         MOPCTR          32, BIDCTR        ; Packets transmitted - deferred
                     0189  1396         MOPCTR          32, BS1CTR        ; Packets transmitted - 1 error
                     018C  1397         MOPCTR          32, BSMCTR        ; Packets transmitted - several errors
                     018F  1398         MOPCTR          16, SFLCTR, MAP   ; Transmit packets aborted
                     0192  1399         MOPCTR          16, RFLCTR, MAP   ; Packets received in error
                     0195  1400         MOPCTR          16, UFDCTR        ; Unrecognized frame destination
                     0198  1401         MOPCTR          16, OVRCTR        ; Receives lost - Internal buffer error
                     019B  1402         MOPCTR          16, LBECTR        ; Receives lost - Local buffer error
                     019E  1403         MOPCTR          16, UBUCTR        ; User buffer unavailable
                     01A1  1404         MOPCTR          16, CDCCTR        ; Transmit collision check failure
                     01A4  1405         MOPCTR                            ; End of table
                     01A6  1406 ;
                     01A6  1407 ; Calculate total size of MOP counter return data buffer
                     01A6  1408 ;
           00000043  01A6  1409         MOP_CTR_SIZE = MOP_CTR_SIZE + 1 + 2 + 8 ; Size of counters + MOP header
           00000051  01A6  1410         MOP_CTR_SIZE = MOP_CTR_SIZE + XBUF_C_HEADER ; Size of buffer + NI header
                     01A6  1411
                     01A6  1412 ;
                     01A6  1413 ; BAD PARAMETER RETURN TABLE
                     01A6  1414 ;
                     01A6  1415 ; First part is validation of Unit against controller. The second part is
                     01A6  1416 ; for validation of shared protocol types.
                     01A6  1417 ;
                     01A6  1418 ; Note that the table is in the REVERSE order from that of the UCB.
                     01A6  1419 ;
                     01A6  1420         ASSUME  UCB$B_XQ_CON EQ UCB$B_XQ_CDBPRM
                     01A6  1421 BAD_PARAM_TBL:
                0456 01A6  1422         .WORD   NMA$C_PCLI_CON
                     01A8  1423
                     01A8  1424         ASSUME  UCB$W_XQ_BSZ EQ UCB$B_XQ_SHRPRM
                     01A8  1425         ASSUME  UCB$B_XQ_PRO EQ UCB$W_XQ_BSZ+2
                     01A8  1426         ASSUME  UCB$B_XQ_PAD EQ UCB$B_XQ_PRO+1
                     01A8  1427         ASSUME  UCB$B_XQ_PRM EQ UCB$B_XQ_PAD+1
                     01A8  1428         ASSUME  UCB$B_XQ_MLT EQ UCB$B_XQ_PRM+1
                     01A8  1429         ASSUME  UCB$B_XQ_DCH EQ UCB$B_XQ_MLT+1
                     01A8  1430         ASSUME  UCB$B_XQ_CDBPRM EQ UCB$B_XQ_DCH+1
                0B1B 01A8  1431         .WORD   NMA$C_PCLI_DCH
```

```
0B19  01AA  1432          .WORD   NMA$C_PCLI_MLT
0B18  01AC  1433          .WORD   NMA$C_PCLI_PRM
0B1A  01AE  1434          .WORD   NMA$C_PCLI_PAD
0458  01B0  1435          .WORD   NMA$C_PCLI_PRO
0B20  01B2  1436          .WORD   NMA$C_PCLI_BSZ
0B20  01B4  1437          .WORD   NMA$C_PCLI_BSZ          ; This parameter is a word (not byte)
```

```
                        .SBTTL  CONTROL_INIT - INITIALIZE DEQNA DEVICE
                ;++
                ; CONTROL_INIT - INITIALIZE DEQNA DEVICE
                ;
                ; Functional description:
                ;
                ; This routine is entered when driver is loaded, system is booted, or
                ; during powerfail recovery.
                ;
                ; Inputs:
                ;
                ;       R4 = Address of the device CSR
                ;       R5 = Address of the device IDB
                ;       R6 = Address of the device DDB
                ;       R8 = Address of the device CRB
                ;
                ;       IPL = FIPL
                ;
                ; Outputs:
                ;
                ;       R4,R5,R8 are preserved
                ;--
                
01B6  1462 CONTROL_INIT::                               ; Initialize the DEQNA
05  01B6  1463         RSB                             ; Return to caller
```

```
                        01B7  1465          .SBTTL  CLONED_UCB - INITIALIZE THE CLONED UCB
                        01B7  1466 ;++
                        01B7  1467 ; CLONED_UCB - INITIALIZE THE CLONED UCB
                        01B7  1468 ;
                        01B7  1469 ; Functional description:
                        01B7  1470 ;
                        01B7  1471 ; This routine is called by the $ASSIGN system service to allow the driver
                        01B7  1472 ; to initialize the cloned UCB. The driver is called with process context.
                        01B7  1473 ;
                        01B7  1474 ; Inputs:
                        01B7  1475 ;
                        01B7  1476 ;       R0 = SS$_NORMAL
                        01B7  1477 ;       R2 = UCB address of cloned UCB
                        01B7  1478 ;       R3 = DDT address
                        01B7  1479 ;       R4 = PCB address
                        01B7  1480 ;       R5 = UCB address of template UCB
                        01B7  1481 ;
                        01B7  1482 ;       IPL = ASTDEL
                        01B7  1483 ;
                        01B7  1484 ; Outputs:
                        01B7  1485 ;
                        01B7  1486 ;       R0 = SS$_NORMAL
                        01B7  1487 ;       R5 = UCB address of cloned UCB
                        01B7  1488 ;       All other registers and IPL are preserved.
                        01B7  1489 ;--
                        01B7  1490
                        01B7  1491 CLONED_UCB::                                 ; Cloned UCB initialization
55   52   D0              01B7  1492          MOVL    R2,R5                      ; Copy UCB address
                        01BA  1493                                              ; Continue in unit_initialization
```

```
                     01BA  1495          .SBTTL  UNIT_INIT - INITIALIZE THE DEQNA UNIT
                     01BA  1496 ;++
                     01BA  1497 ; UNIT_INIT - INITIALIZE THE DEQNA UNIT
                     01BA  1498 ;
                     01BA  1499 ; Functional description:
                     01BA  1500 ;
                     01BA  1501 ; This routine is called at system startup, during driver loading and
                     01BA  1502 ; during powerfail recovery to initialize the DEQNA unit and its UCB.
                     01BA  1503 ; The UCB is initialized and if power has failed, the device is forced
                     01BA  1504 ; to shutdown.
                     01BA  1505 ;
                     01BA  1506 ; Inputs:
                     01BA  1507 ;
                     01BA  1508 ;       R4 = CSR address
                     01BA  1509 ;       R5 = UCB address
                     01BA  1510 ;
                     01BA  1511 ;       IPL = FIPL
                     01BA  1512 ;
                     01BA  1513 ; Outputs:
                     01BA  1514 ;
                     01BA  1515 ;       None.
                     01BA  1516 ;--
                     01BA  1517
                     01BA  1518 UNIT_INIT::                          ; Initialize a DEQNA unit
                     01BA  1519 ;;&&    JSB     G^INI$BRK           ;*** TEMP ***
                     01BA  1520
          3F   BB    01BA  1521         PUSHR   #^M<R0,R1,R2,R3,R4,R5>  ; Save all regs
19 64 A5  05   E0    01BC  1522         BBS     #UCB$V_POWER,UCB$W_STS(R5),15$ ; Br if powerfail
                     01C1  1523
                     01C1  1524 ;
                     01C1  1525 ; Initialize UCB queue listheads, and the pointer (within the NI device
                     01C1  1526 ; dependent UCB extension, UCB$L_NI_MLTPTR) to the multicast address table for
                     01C1  1527 ; this protocol type.
                     01C1  1528 ;
                     01C1  1529
     00E7 C5   9E    01C1  1530         MOVAB   UCB$G_XQ_MULTI(R5),-    ; Initialize the pointer to this
     0094 C5         01C5  1531                 UCB$L_NI_MLTPTR(R5)     ; protocol's multicast address table
                     01C8  1532
     50   04   D0    01C8  1533         MOVL    #UCB$C_XQ_QUEUES,R0     ; Get number of queue listheads in UCB
51   0098 C5   9E    01CB  1534         MOVAB   UCB$Q_XQ_QUEUES(R5),R1  ; Get address of queue listheads
     81   61   DE    01D0  1535 10$:    MOVAL   (R1),(R1)+              ; Set forward link pointer
  81   FC A1   D0    01D3  1536         MOVL    -4(R1),(R1)+            ; Set backward link pointer
       F6 50   F5    01D7  1537         SOBGTR  R0,10$                  ; Loop if more listheads
                     01DA  1538
       54 A5   B5    01DA  1539 15$:    TSTW    UCB$W_UNIT(R5)          ; Is this unit 0?
          15   13    01DD  1540         BEQL    17$                     ; Br if yes - leave TEMPLATE bit on
     0082 C5   B4    01DF  1541         CLRW    UCB$W_ERRCNT(R5)        ; Only unit #0 may list errors
                     01E3  1542 ;
                     01E3  1543 ; We must find the address of unit 0, so we can check if the QNA is ONLINE.
                     01E3  1544 ; If the QNA is OFFLINE, then we mark each UCB as being OFFLINE.
                     01E3  1545 ;
50   28 A5   D0    01E3  1546         MOVL    UCB$L_DDB(R5),R0        ; Get address of QNA DDB
50   04 A0   D0    01E7  1547         MOVL    DDB$L_UCB(R0),R0        ; Get address of UNIT 0 UCB
          04   E0    01EB  1548         BBS     #UCB$V_ONLINE,-         ; Br if QNA is ONLINE
  04 64 A0         01ED  1549                 UCB$W_STS(R0),17$       ;
          10   AA    01F0  1550         BICW    #UCB$M_ONLINE,-         ; Else, mark new unit as OFFLINE
       64 A5         01F2  1551                 UCB$W_STS(R5)           ;
```

```
                     01F4  1552 17$:    ;
                     01F4  1553         ; For u-VAX I, we will have to allocate a physically contiguous buffer
                     01F4  1554         ; area for perfoming I/O on the QNA.
                     01F4  1555         ;
                     01F4  1556         CPUDISP <<790,60$>,-
                     01F4  1557                 <780,60$>,-
                     01F4  1558                 <750,60$>,-
                     01F4  1559                 <730,60$>,-
                     01F4  1560                 <UV1,20$>>                  ; For u-VAX I, allocate buffer area
                     020E  1561                                             ; For all others, skip buffer area
                     020E  1562
   51    24 A5   D0  020E  1563 20$:    MOVL    UCB$L_CRB(R5),R1            ; Get CRB address
   54    10 A1   D0  0212  1564         MOVL    CRB$L_AUXSTRUC(R1),R4       ; Get CDB address
         0A      12  0216  1565         BNEQ    23$                         ; Br if present
       1D66      30  0218  1566         BSBW    ALLOC_CDB                   ; Else, try to allocate a CDB
      23 50      E9  021B  1567         BLBC    R0,60$                      ; Br if error
   54    10 A1   D0  021E  1568         MOVL    CRB$L_AUXSTRUC(R1),R4       ; Get CDB address
    0210 C4      D5  0222  1569 23$:    TSTL    CDB_L_UV1BUF(R4)            ; Is the buffer area allocated?
         19      12  0226  1570         BNEQ    60$                         ; Br if yes, continue
                     0228  1571
          00002400   0228  1572 UV1_BUFFER_LENGTH = <UV1_BUFFER_AREA + 511> & <^C511>   ; Round to a page
          00000012   0228  1573 UV1_BUFFER_PAGES  = UV1_BUFFER_LENGTH / 512             ; Number of pages
                     0228  1574
      51    12   3C  0228  1575         MOVZWL  #UV1_BUFFER_PAGES,R1        ; Number of pages to allocate
00000000'GF    16   0228  1576         JSB     G^EXE$ALOPH?CNTG            ; Allocate physically-contiguous memory
      0D 50   E9  0231  1577         BLBC    R0,60$                      ; Skip ahead on error
   54    24 A5   D0  0234  1578         MOVL    UCB$L_CRB(R5),R4            ; Get CRB address
   54    10 A4   D0  0238  1579         MOVL    CRB$L_AUXSTRUC(R4),R4       ; Get CDB address
 0210 C4    52   D0  023C  1580         MOVL    R2,CDB_L_UV1BUF(R4)         ; Save buffer area address
                     0241  1581
      50    24   9A  0241  1582 60$:    MOVZBL  #3*MAX_C_MLT,R0             ; Get size of multicast list in words
   51    00E7 C5  9E  0244  1583         MOVAB   UCB$G_XQ_MULTI(R5),R1       ; Get address of multicast list
         81      B4  0249  1584 70$:    CLRW    (R1)+                       ; Init multicast table
         FB 50   F5  024B  1585         SOBGTR  R0,70$                      ; Loop if more
   54    24 A5   D0  024E  1586         MOVL    UCB$L_CRB(R5),R4            ; Get CRB address
   54    10 A4   D0  0252  1587         MOVL    CRB$L_AUXSTRUC(R4),R4       ; Get CDB address
         06      12  0256  1588         BNEQ    80$                         ; Br if present
       1D26      30  0258  1589         BSBW    ALLOC_CDB                   ; Else, try to allocate a CDB
      0F 50      E9  025B  1590         BLBC    R0,90$                      ; Br if no error
 0A 64 A5        E1  025E  1591 80$:    BBC     #UCB$V_POWER,UCB$W_STS(R5),90$ ; Br if not powerfail
53    00024000 8F   D0  0263  1592         MOVL    #<XQ_SOFT_M_POWERAT6>!- ; Indicate cause of error
                     026A  1593                 XQ_CSR_M_ERR,R3
       1456      30  026A  1594         BSBW    SCHED_FORK                  ; Schedule fork process
         3F      BA  026D  1595 90$:    POPR    #^M<R0,R1,R2,R3,R4,R5>      ; Restore regs
                 05  026F  1596         RSB                                 ; Done
```

```
                     0270  1598          .SBTTL  FFI_INIT - FFI INTERFACE INITIALIZATION ROUTINE
                     0270  1599 ;++
                     0270  1600 ; FFI_INIT - FFI INTERFACE INITIALIZATION ROUTINE
                     0270  1601 ;
                     0270  1602 ; Functional description:
                     0270  1603 ;
                     0270  1604 ; This routine initializes the FFI interface. Currently the UCB must
                     0270  1605 ; have been initialized prior to calling this routine, in the future
                     0270  1606 ; this routine may have to initialize the UCB and DEQNA. Therefore,
                     0270  1607 ; there may be a fork involved in the call to this routine.
                     0270  1608 ;
                     0270  1609 ; Inputs:
                     0270  1610 ;
                     0270  1611 ;       R3 = Address of quadword descriptor for parameter buffer
                     0270  1612 ;       R4 = FFI block address
                     0270  1613 ;
                     0270  1614 ;       IPL = SYNCH
                     0270  1615 ;
                     0270  1616 ; Outputs:
                     0270  1617 ;
                     0270  1618 ;       R0 = Status of request
                     0270  1619 ;       All other registers are preserved.
                     0270  1620 ;
                     0270  1621 ;--
                     0270  1622          ASSUME  IPL$_SYNCH EQ IPL$_XQ_FIPL
                     0270  1623 FFI_INIT::
            3E   BB  0270  1624          PUSHR   #^M<R1,R2,R3,R4,R5>     ; Save registers
            50   D4  0272  1625          CLRL    R0                      ; Assume failure
     55  34   A4  D0  0274  1626          MOVL    FFI$L_DL_UCB(R4),R5     ; Get UCB address
            04   E1  0278  1627          BBC     #UCB$V_XQ_RUN,-         ; Br if device not ready
       0E  68   A5      027A  1628                  UCB$W_DEVSTS(R5),90$
        034C'  CF   9E  027D  1629          MOVAB   W^XMT_FFI_START,-       ; Return address of XMIT routine
            10   A4      0281  1630                  FFI$L_XMIT(R4)
 018D   C5       54  D0  0283  1631          MOVL    R4,UCB$L_XQ_FFI(R5)     ; Save FFI address
            50   01  9A  0288  1632          MOVZBL  #1,R0                   ; Return success
            3E   BA  028B  1633 90$:        POPR    #^M<R1,R2,R3,R4,R5>     ; Restore registers
                 05  028D  1634          RSB                             ; Return to caller
                     028E  1635
```

```
                                        .SBTTL  XMT_FDT - TRANSMIT I/O OPERATION FDT ROUTINE
                                ;++
                                ; XMT_FDT - TRANSMIT I/O OPERATION FDT ROUTINE
                                ;
                                ; Functional description:
                                ;
                                ; This routine sets up the internal function code for transmit and
                                ; transfers control to the exec buffered I/O write FDT routine.
                                ;
                                ; The QIO parameters for WRITES are:
                                ;
                                ;       P1 = Address of the data buffer
                                ;       P2 = Size of the data buffer
                                ;       P5 = Address of buffer containing the destination address
                                ;
                                ;  ** The driver can never do direct I/O on XMIT requests, because  **
                                ; *** the QNA buffer address cannot begin on an odd byte boundary.  ***
                                ;  **    Also, the FAST interface cannot operate on DIRECT I/O.     **
                                ;
                                ;
                                ; Inputs:
                                ;
                                ;       R3 = IRP address
                                ;       R4 = PCB address
                                ;       R5 = UCB address
                                ;       R6 = CCB address
                                ;       R7 = FUNCTION CODE
                                ;
                                ;       IPL = ASTDEL
                                ;
                                ; Outputs:
                                ;
                                ;       R0-R2,R8,R9 are destroyed.
                                ;
                                ;--
                                ABORTIO_BR:                                     ; Long branch to ABORTIO
00B5   31                       BRW     ABORTIO                                 ; Abort the I/O request
                                ;
                                XMT_FDT::                                       ; Transmit FDT routine
   40 A3   7C                   CLRQ    IRP$Q_STATION(R3)                       ; Zero the destination address
00CA C5   B0                    MOVW    UCB$W_XQ_PROTYP(R5),-                   ; Assume we are a non-promiscuous user
   3A A3                        IRP$W_XQ_PROTYP(R3)                             ;
51    10 AC   D0                MOVL    P5(AP),R1                               ; Get address of destination address
      05   12                   BNEQ    10$                                     ; Br if given
                                ;
                                ; If the user is in shared mode, then he does not have to supply a destination
                                ; address with each transmit operation. The destination address will be gotten
                                ; from the SHR data structure.
                                ;
      03   E0                   BBS     #UCB$V_XQ_SHARE,-                       ; Br if shared user
12 68 A5                        UCB$W_DEVSTS(R5),20$                            ;
   50   0C   9A                MOVZBL  S^#SS$_ACCVIO,R0                        ; Assume access violation
                                IFNORD  #6,(R1),ABORTIO_BR                      ; Check access to buffer
40 A3   61   D0                 MOVL    (R1),IRP$Q_STATION(R3)                  ; Save destination address
44 A3   04 A1   B0              MOVW    4(R1),IRP$Q_STATION+4(R3)               ;     ...
                                ;
                                        ASSUME  NMA$C_STATE_ON EQ 0
```

```
                          02B7  1694          ASSUME  NMA$C_STATE_OFF EQ 1
                          02B7  1695
          0D 00DA C5   E8 02B7  1696 20$:     BLBS    UCB$B_XQ_PRM(R5),30$     ; Br if user is not promiscuous
             51   06   C0 02BC  1697          ADDL    #6,R1                    ; Point to protocol type
                          02BF  1698          IFNORD  #2,(R1),ABORTIO_BR       ; Check access to buffer
          3A A3   61   B0 02C5  1699          MOVW    (R1),IRP$W_XQ_PROTYP(R3) ; Get protocol type from user P5 buffer
             50   14   9A 02C9  1700 30$:     MOVZBL  S^#SS$_BADPARAM,R0       ; Assume bad parameters
             58   6C   D0 02CC  1701          MOVL    P1(AP),R8                ; Get starting address of user buffer
          59    04 AC   3C 02CF  1702          MOVZWL  P2(AP),R9                ; Get length of user buffer
                   71   13 02D3  1703          BEQL    ABORTIO                  ; Br if zero length buffer
             50   58   7D 02D5  1704          MOVQ    R8,R0                    ; Retrieve buffer parameters
       00000000'GF   16   02D8  1705          JSB     G^EXE$WRITECHK           ; Check accessibility of user buffer
                          02DE  1706                                           ; (No return on NO ACCESS)
                          02DE  1707                                           ; Returns IRP$W_BCNT
             2E   51   B1 02DE  1708          CMPW    R1,#MIN_PKT_SIZE         ; Is buffer at least minimum?
                   03   1E 02E1  1709          BGEQU   50$                      ; Br if yes, okay
             51   2E   3C 02E3  1710          MOVZWL  #MIN_PKT_SIZE,R1         ; Else, allocate minimum sized packet
       51 00000048 8F   C0 02E6  1711 50$:     ADDL2   #CXB$C_HEADER,R1         ; Calculate length of buffer needed
                   38   BB 02ED  1712          PUSHR   #^M<R3,R4,R5>            ; Save registers
       00000000'GF   16   02EF  1713          JSB     G^EXE$BUFFRQUOTA         ; Check if process has sufficient quota
                4C 50   E9 02F5  1714          BLBC    R0,90$                   ; Br if quota check failure
       00000000'GF   16   02F8  1715          JSB     G^EXE$ALLOCBUF           ; Allocate CXB buffer for output
                43 50   E9 02FE  1716          BLBC    R0,90$                   ; If LBC allocation failure
             53   6E   D0 0301  1717          MOVL    (SP),R3                  ; Retrieve address of IRP
          50    0080 C4   D0 0304  1718          MOVL    PCB$L_JIB(R4),R0         ; Get JIB address
            20 A0   51   C2 0309  1719          SUBL    R1,JIB$L_BYTCNT(R0)      ; Adjust buffered I/O quota
            30 A3   51   B0 030D  1720          MOVW    R1,IRP$W_BOFF(R3)        ; Set number of bytes charged to quota
            2C A3   52   D0 0311  1721          MOVL    R2,IRP$L_SVAPTE(R3)      ; Save CXB address in IRP
                    52   DD 0315  1722          PUSHL   R2                       ; Save pointer to CXB
                          0317  1723
                          0317  1724          ASSUME  CXB$L_FL EQ 0
                          0317  1725          ASSUME  CXB$L_BL EQ CXB$L_FL+4
                    82   7C 0317  1726          CLRQ    (R2)+                    ; Clear link cells
                          0319  1727
                          0319  1728          ASSUME  CXB$W_SIZE EQ CXB$L_BL+4
                 82 51   B0 0319  1729          MOVW    R1,(R2)+                 ; Set size of structure
                          031C  1730
                          031C  1731          ASSUME  CXB$B_TYPE EQ CXB$W_SIZE+2
                          031C  1732          ASSUME  CXB$B_CODE EQ CXB$B_TYPE+1
                 82 1B   9B 031C  1733          MOVZBW  #DYN$C_CXB,(R2)+         ; Set structure type
                          031F  1734
                 52 6E   D0 031F  1735          MOVL    (SP),R2                  ; Get back CXB address
                          0322  1736
                          0322  1737          ASSUME  CXB$C_HEADER EQ CXB$T_T_DATA+XQ_C_HEADER
               18 A2 3A   B0 0322  1738          MOVW    #CXB$T_T_DATA,CXB$W_BOFF(R2) ; Setup offset to start of data
             52    48 A2 9E 0326  1739          MOVAB   CXB$C_HEADER(R2),R2      ; Get address of data portion of buffer
                          032A  1740
          62    68   59   28 032A  1741          MOVC3   R9,(R8),(R2)             ; Move data to system buffer
                    3C   BA 032E  1742          POPR    #^M<R2,R3,R4,R5>         ; Restore registers
                          0330  1743          SETIPL  UCB$B_FIPL(R5)           ; Sync access to UCB
                    53   DD 0334  1744          PUSHL   R3                       ; Save IRP address
                    57   10 0336  1745          BSBB    XMT_START                ; Do common processing
                    53 8ED0 0338  1746          POPL    R3                       ; Restore IRP address
                 08 50   E9 033B  1747          BLBC    R0,ABORTIO               ; Br if error in processing request
       00000000'GF   17   033E  1748          JMP     G^EXE$QIORETURN          ; Exit QIO service to await completion
                          0344  1749
                    38   BA 0344  1750 90$:     POPR    #^M<R3,R4,R5>            ; Restore registers
```

```
00000000'GF   17  0346  1751 ABORTIO:JMP     G^EXE$ABORTIO           ; Abort the I/O request
```

```
                     034C  1753          .SBTTL  XMT_FFI_START - START FAST INTERFACE TRANSMIT OPERATION
                     034C  1754 ;++
                     034C  1755 ; XMT_FFI_START - START FAST INTERFACE TRANSMIT OPERATION
                     034C  1756 ;
                     034C  1757 ; Functional description:
                     034C  1758 ;
                     034C  1759 ; This routine is called to start a transmit operation. If the QNA is running
                     034C  1760 ; then the request is given to the xmit wait queue for the QNA. If there is
                     034C  1761 ; a free entry in the transmit ring and there are sufficient map registers to
                     034C  1762 ; map the buffer then the request is given to the QNA immediately, else the
                     034C  1763 ; request is left on the xmit wait queue until another request completes.
                     034C  1764 ;
                     034C  1765 ;
                     034C  1766 ; Inputs:
                     034C  1767 ;
                     034C  1768 ;       R3 = CXB address
                     034C  1769 ;       R4 = FFI address
                     034C  1770 ;
                     034C  1771 ;       IPL = SYNCH (same as FIPL)
                     034C  1772 ;
                     034C  1773 ; Outputs:
                     034C  1774 ;
                     034C  1775 ;       R0,R3 are destroyed.
                     034C  1776 ;       All other registers are preserved.
                     034C  1777 ;
                     034C  1778 ;       If the request cannot be queued, then the FFI$L_XMIT_DONE entry is
                     034C  1779 ;       called immediately with the following:
                     034C  1780 ;
                     034C  1781 ;               R0 = Status of transmit request
                     034C  1782 ;               R3 = CXB address
                     034C  1783 ;               R4 = FFI address
                     034C  1784 ;
                     034C  1785 ;--
                     034C  1786          ASSUME  IPL$_SYNCH EQ IPL$_XQ_FIPL
                     034C  1787 XMT_FFI_START::                          ; Start FAST interface transmit request
          36   BB    034C  1788          PUSHR   #^M<R1,R2,R4,R5>        ; Save registers
    55    34 A4   D0  034E  1789          MOVL    FFI$L_DL_UCB(R4),R5     ; Get UCB address
                     0352  1790          ;
                     0352  1791          ;  For the FFI Interface, we will save the UCB address
                     0352  1792          ;
    24 A3   55   D0  0352  1793          MOVL    R5,CXB$L_T_UCB(R3)      ; Save UCB address
                     0356  1794          ASSUME  CXB$L_T_UCB  EQ  CXB$L_T_IRP
    24 A3   01   88  0356  1795          BISB    #1,CXB$L_T_UCB(R3)      ; ...indicate this is a UCB address
                     035A  1796
    18 A3   0E   A2  035A  1797          SUBW    #XQ_C_HEADER,CXB$W_BOFF(R3) ; Back up offset for Ethernet header
                     035E  1798
                     035E  1799          ASSUME  NMA$C_STATE_ON EQ 0
                     035E  1800          ASSUME  NMA$C_STATE_OFF EQ 1
  04 00D9 C5   E8    035E  1801          BLBS    UCB$B_XQ_PAD(R5),20$    ; Br if padding is disabled
  18 A3     02   A2  0363  1802          SUBW    #XQ_C_CNTSIZ,CXB$W_BOFF(R3) ; Else, skip length field of buffer
  51   18 A3   3C  0367  1803 20$:       MOVZWL  CXB$W_BOFF(R3),R1       ; Get offset to start of data
52    53   51   C1  036B  1804          ADDL3   R1,R3,R2                ; Set R2 to start of header
                     036F  1805
                     036F  1806          ASSUME  XBUF_G_SRC EQ XBUF_G_DEST+6
        28 A3   7D  036F  1807          MOVQ    CXB$Q_STATION(R3),-     ; Store destination address
           62       0372  1808            XBUF_G_DEST(R2)
                     0373  1809          ASSUME  XBUF_W_TYPE EQ XBUF_G_SRC+6
```

```
                00DE C5   7D   0373  1810          MOVQ    UCB$G_XQ_PHA(R5),-      ; Store our source address
                  06 A2         0377  1811                  XBUF_G_SRC(R2)
                00CA C5   B0   0379  1812          MOVW    UCB$W_XQ_PROTYP(R5),-   ; Set PROTOCOL TYPE
                  0C A2         037D  1813                  XBUF_W_TYPE(R2)
                     56   10   037F  1814          BSBB    XMT_INITIATE            ; Try to start transmit
                  08 50   E8   0381  1815          BLBS    R0,50$                  ; Br if success
          54    018D C5   D0   0384  1816          MOVL    UCB$L_XQ_FFI(R5),R4     ; Else, get back FFI address
                  14 B4   16   0389  1817          JSB     @FFI$C_XMIT_DONE(R4)    ; Complete request in error
                     36   BA   038C  1818 50$:      POPR    #^M<R1,R2,R4,R5>        ; Restore registers
                          05   038E  1819          RSB                             ; Return to caller
```

```
                                        .SBTTL  XMT_START - START TRANSMIT OPERATION
                                ;++
                                ; XMT_START - START TRANSMIT OPERATION
                                ;
                                ; Functional description:
                                ;
                                ; This routine is called to start a transmit operation. If the QNA is running
                                ; then the request is given to the xmit wait queue for the QNA. If there is
                                ; a free entry in the transmit ring and there are sufficient map registers to
                                ; map the buffer then the request is given to the QNA immediately, else the
                                ; request is left on the xmit wait queue until another request completes.
                                ;
                                ;  ** The driver can never do direct I/O on XMIT requests, because  **
                                ; *** the QNA buffer address cannot begin on an odd byte boundary.  ***
                                ;  **    Also, the FAST interface cannot operate on DIRECT I/O.     **
                                ;
                                ; Inputs:
                                ;
                                ;       R2 = CXB address
                                ;       R3 = IRP address
                                ;       R5 = UCB address
                                ;
                                ;       IPL = FIPL
                                ;
                                ; Outputs:
                                ;
                                ;       R0 = Status of transmit request
                                ;       R1,R2,R4 are destroyed.
                                ;
                                ;--
                                        .ENABL  LSB
                                XMT_START::                             ; Start transmit operation
                 04     E0      038F    1855            BBS     #UCB$V_XQ_RUN,-         ; Br if unit is in RUN mode
             10 68 A5            0391    1856                    UCB$W_DEVSTS(R5),30$
                                 0394    1857
          50   20D4 8F     3C    0394    1858 10$:       MOVZWL  #SS$_DEVINACT,R0        ; Assume unit not started yet
                 0E     E1      0399    1859            BBC     #UCB$V_XQ_INTERLOCK,-   ; Br if unit is not re-starting
             05 68 A5            039B    1860                    UCB$W_DEVSTS(R5),20$    ;  on it's own.
          50   02D4 8F     3C    039E    1861            MOVZWL  #SS$_OPINCOMPL,R0       ; Else, return different error code
                 05     03A3    1862 20$:       RSB                             ; Okay to leave now
                               03A4    1863
            24 A2    53     D0    03A4    1864 30$:       MOVL    R3,CXB$L_T_IRP(R2)      ; Save IRP address
                 0094 C3     D4    03A8    1865            CLRL    IRP$L_XQ_SETUP(R3)      ; Indicate no SETUP buffer present
        1A A2    32 A3     B0    03AC    1866            MOVW    IRP$W_BCNT(R3),CXB$W_BCNT(R2) ; Set BCNT in CXB
                 54     53     D0    03B1    1867            MOVL    R3,R4                   ; Copy IRP address
                 53     52     D0    03B4    1868            MOVL    R2,R3                   ; Copy CXB address
                               03B7    1869
                               03B7    1870            ASSUME  NMA$C_STATE_ON EQ 0
                               03B7    1871            ASSUME  NMA$C_STATE_OFF EQ 1
            04 00D9 C5     E8    03B7    1872            BLBS    UCB$B_XQ_PAD(R5),40$    ; Br if padding is disabled
            18 A3    02     A2    03BC    1873            SUBW    #XQ_C_CNTSIZ,CXB$W_BOFF(R3) ; Else, skip length field of buffer
          51   18 A3     3C    03C0    1874 40$:       MOVZWL  CXB$W_BOFF(R3),R1       ; Get offset to start of data
        52   53     51     C1    03C4    1875            ADDL3   R1,R3,R2                ; Set R2 to start of header
                               03C8    1876            ASSUME  XBUF_G_SRC EQ XBUF_G_DEST+6
                 40 A4     7D    03C8    1877            MOVQ    IRP$Q_STATION(R4),-     ; Store destination address
```

```
                     62        03CB  1878                    XBUF_G_DEST(R2)
                               03CC  1879            ASSUME  XBUF_W_TYPE EQ XBUF_G_SRC+6
                00DE C5   7D   03CC  1880            MOVQ    UCB$G_XQ_PHA(R5),-      ; Store our source address
                  06 A2        03D0  1881                    XBUF_G_SRC(R2)
                  3A A4   B0   03D2  1882            MOVW    IRP$W_XQ_PROTYP(R4),-   ; Store PROTOCOL TYPE
                  0C A2        03D5  1883                    XBUF_W_TYPE(R2)
                               03D7  1884
                               03D7  1885 XMT_INITIATE::                              ; FAST Interface entry point (FFI)
                               03D7  1886                ;
                               03D7  1887                ; Inputs:
                               03D7  1888                ;       R5 = UCB address
                               03D7  1889                ;       R3 = CXB address
                               03D7  1890                ;       R2 = Start address for Ethernet header
                               03D7  1891                ;
            20 A3   01   90    03D7  1892            MOVB    #XQ_FC_V_XMIT,CXB$B_XQ_FUNC(R3) ; Set function request in CXB
                               03DB  1893
                               03DB  1894            ASSUME  NMA$C_STATE_ON EQ 0
                               03DB  1895            ASSUME  NMA$C_STATE_OFF EQ 1
            09 00D9 C5   E8    03DB  1896            BLBS    UCB$B_XQ_PAD(R5),60$    ; Br if padding is disabled
                               03E0  1897 ;
                               03E0  1898 ;  PADDING IS ENABLED:
                               03E0  1899 ;       Adjust byte count to include size field and store count field.
                               03E0  1900 ;
                1A A3   B0     03E0  1901            MOVW    CXB$W_BCNT(R3),-        ; Else, store size of data-only
                  0E A2        03E3  1902                    XBUF_W_SIZE(R2)         ;  portion of buffer in message
            1A A3   02   A0    03E5  1903            ADDW    #XQ_C_CNTSIZ,CXB$W_BCNT(R3) ; And account for count field
                               03E9  1904 ;
                               03E9  1905 ; Allow buffer size up to Ethernet max buffer size for transmit operations.
                               03E9  1906 ;
        05DC 8F    1A A3   B1  03E9  1907 60$:    CMPW    CXB$W_BCNT(R3),#MAX_PKT_SIZE ; Is buffer size bigger than
                               03EF  1908                                            ;  largest Ethernet buffer allowed?
                      06   1B  03EF  1909            BLEQU   90$                     ; Br if no
           50    034C 8F   3C  03F1  1910 80$:    MOVZWL  #SS$_IVBUFLEN,R0        ; Assume bad buffer length
                      05   05  03F6  1911            RSB                             ; ELSE, leave now
                               03F7  1912
              54    24 A5   D0  03F7  1913 90$:    MOVL    UCB$L_CRB(R5),R4        ; Get CRB address
              54    10 A4   D0  03FB  1914            MOVL    CRB$L_AUXSTRUC(R4),R4   ; Get CDB address
                      01   E1  03FF  1915            BBC     #CDB_STS_V_RUN,-        ; Br if QNA not running
            1B 024A C4         0401  1916                    CDB_B_STS(R4),100$      ;
                      0E   A0  0405  1917            ADDW    #XQ_C_HEADER,-          ; Adjust byte count
                1A A3          0407  1918                     CXB$W_BCNT(R3)         ;  for header info
                               0409  1919 ;
                               0409  1920 ; If running in the SHARED-LIMITED mode, then we must use the destination
                               0409  1921 ; address from the SHR_ data structure. Unless the given destination address
                               0409  1922 ; is a multicast address. For the SHARED-DEFAULT user, we must make sure that
                               0409  1923 ; destination address given is unique!
                               0409  1924 ;
                      03   E1  0409  1925            BBC     #UCB$V_XQ_SHARE,-       ; Br if NOT a shared user
                4E 68 A5       040B  1926                    UCB$W_DEVSTS(R5),NO_SHR ;
                               040E  1927 ;
                               040E  1928 ; Try to find a match on PID/CHAN. Returns pointer in R1
                               040E  1929 ;
                               040E  1930            ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
                0E 24 A3   E8  040E  1931            BLBS    CXB$L_T_IRP(R3),100$    ; Br if FFI user, return failure
                      53   DD  0412  1932            PUSHL   R3                      ; Else, save CXB address
              53    24 A3   D0  0414  1933            MOVL    CXB$L_T_IRP(R3),R3      ; Get IRP address
                    02F0   30  0418  1934            BSBW    MATCH_SHR               ; Try to find the SHR data structure
```

```
                     06   13   041B  1935          BEQL    110$                        ; Br if match
                     53 8ED0   041D  1936          POPL    R3                          ; Restore CXB address
                   FF71   31   0420  1937 100$:     BRW     10$                         ; Else, error
                          0423  1938
                     53 8ED0   0423  1939 110$:     POPL    R3                          ; Restore CXB address
               51  00C4 C5     D1   0426  1940          CMPL    UCB$L_XQ_DEFUSR(R5),R1      ; Is this the default user?
                     0E   13   042B  1941          BEQL    SHR_DEF                     ; Br if yes
                          042D  1942 ;
                          042D  1943 ; This is a SHARED-LIMITED user.
                          042D  1944 ;
                   2C 62   E8   042D  1945          BLBS    XBUF_G_DEST(R2),NO_SHR      ; Br if multicast address
                   12 A1   D0   0430  1946          MOVL    SHR_G_DEST(R1),-            ; Else, get destination from SHR struct.
                      62        0433  1947                    XBUF_G_DEST(R2)             ;       ...
                   16 A1   B0   0434  1948          MOVW    SHR_G_DEST+4(R1),-          ;       ...
                   04 A2        0437  1949                    XBUF_G_DEST+4(R2)           ;       ...
                     21   11   0439  1950          BRB     NO_SHR                      ; Continue in common code
                          043B  1951          .DSABL  LSB
                          043B  1952 ;
                          043B  1953 ; This is a SHARED-DEFAULT user.
                          043B  1954 ;
               50  0098 C5     9E   043B  1955 SHR_DEF:MOVAB   UCB$Q_XQ_SHARE(R5),R0    ; Get address of SHR listhead
                     51   50   D0   0440  1956          MOVL    R0,R1                       ; Save address of start of listhead
                     51   61   D0   0443  1957 10$:     MOVL    SHR_L_QFL(R1),R1            ; Get address of next in list
                     51   50   D1   0446  1958          CMPL    R0,R1                       ; Back at start of list?
                          11   13   0449  1959          BEQL    NO_SHR                      ; Br if yes, destination is unique
                          62   D1   044B  1960          CMPL    XBUF_G_DEST(R2),-           ; Address match?
                      12 A1        044D  1961                    SHR_G_DEST(R1)              ;       ...
                          F2   12   044F  1962          BNEQ    10$                         ; Br if no - check next in list
                      04 A2   B1   0451  1963          CMPW    XBUF_G_DEST+4(R2),-         ; Hi order of address match?
                      16 A1        0454  1964                    SHR_G_DEST+4(R1)            ;       ...
                          EB   12   0456  1965          BNEQ    10$                         ; Br if not - check next in list
                     50   14   3C   0458  1966          MOVZWL  #SS$_BADPARAM,R0            ; Else, bad parameter code
                          05   045B  1967          RSB                                 ; Return to caller
                          045C  1968
                          045C  1969 NO_SHR:
                          045C  1970          ;
                          045C  1971          ; Do accounting for MULTICAST here. NOTE: this should really
                          045C  1972          ; be done upon completion of request, but it is done here to
                          045C  1973          ; save extra work to check for multicast in the completion
                          045C  1974          ; section.
                          045C  1975          ;
                     1A 62   E9   045C  1976          BLBC    XBUF_G_DEST(R2),20$         ; Br if NOT multicast address
                          045F  1977          INCC    CDB_L_MBSCTR(R4)            ; Count multicast blocks sent
               50    1A A3   3C   0469  1978          MOVZWL  CXB$W_BCNT(R3),R0           ; Get BCNT
                          046D  1979          CNTR    R0,CDB_L_MSNCTR(R4),L       ; Count multicast bytes sent
                          0479  1980 20$:
                          0479  1981 ;
                          0479  1982 ; If we are running in point-to-point mode, then queue xmit on wait queue
                          0479  1983 ; if we are waiting for run!
                          0479  1984 ;
                        00   91   0479  1985          CMPB    #NMA$C_LINPR_POI,-          ;% Are we in PT-TO-PT mode?
                   00D8 C5        047B  1986                    UCB$B_XQ_PRO(R5)            ;%
                          0E   12   047E  1987          BNEQ    40$                         ;% Br if not
                          06   E1   0480  1988          BBC     #UCB$V_XQ_STACK,-           ;% Br if not in stack wait state
                09 68 A5        0482  1989                    UCB$W_DEVSTS(R5),40$        ;%
           00B4 D5      63   0E   0485  1990          INSQUE  (R3),@UCB$Q_XQ_XMTREQ+4(R5);% Else, insert request on wait queue
                     50   01   9A   048A  1991          MOVZBL  #1,R0                       ;% Return success
```

```
                     05  048D  1992          RSB                                 ;% Return to caller
                         048E  1993 ;
                         048E  1994 ; Insert request on CDB transmit request queue and check if transmit
                         048E  1995 ; can proceed.
                         048E  1996 ;
          00E0 D4    63  0E  048E  1997 40$:     INSQUE  (R3),@CDB_Q_XMTREQ+4(R4) ; Insert at end of xmit queue
                         0493  1998
                         0493  1999          .ENABL  LSB
                         0493  2000 XMT_ALT_START::                              ; Alternate start for xmit
                         0493  2001          PUSHQ   R6                          ; Save R6,R7
            56   24 A5   D0  0496  2002          MOVL    UCB$L_CRB(R5),R6        ; Get CRB address
            54   10 A6   D0  049A  2003          MOVL    CRB$L_AUXSTRUC(R6),R4   ; Get CDB address
                         049E  2004 ;
                         049E  2005 ; Skip MAP register useage if u-VAX I. Also use different number of slots.
                         049E  2006 ;
                         049E  2007          CPUDISP <<790,10$>,-
                         049E  2008                   <780,10$>,-
                         049E  2009                   <750,10$>,-
                         049E  2010                   <730,10$>,-
                         049E  2011                   <UV1,XMT_UV1>>
                         04B8  2012
                         04B8  2013 10$:     ASSUME  MAX_C_XMT LE 8
     57   1B A4   03  00  EB  04B8  2014          FFC     #0,#MAX_C_XMT-1,CDB_B_XMTMAP(R4),R7 ; Find a free transmit slot
                     42  13  04BE  2015          BEQL    20$                     ; Br if none free
                         04C0  2016 ;
                         04C0  2017 ; Move CXB info into UCB
                         04C0  2018 ;
            53   00DC D4    0F  04C0  2019          REMQUE  @CDB_Q_XMTREQ(R4),R3    ; Get oldest xmit request
                     3B  1D  04C5  2020          BVS     20$                     ; Br if none
          7E A5    1A A3   B0  04C7  2021          MOVW    CXB$W_BCNT(R3),UCB$W_BCNT(R5) ; Set byte count
            52     18 A3   3C  04CC  2022          MOVZWL  CXB$W_BOFF(R3),R2       ; Get offset to start of data
                52   53   C0  04D0  2023          ADDL    R3,R2                   ; Compute buffer virtual address
                FE00 8F   AB  04D3  2024          BICW3   #^C<VA$M_BYTE>,-        ; Get buffer offset
             7C A5    52      04D7  2025                  R2,UCB$W_BOFF(R5)       ;
                         04DA  2026 ;
                         04DA  2027 ; Convert virtual address to physical PTE address
                         04DA  2028 ;
                     09  EF  04DA  2029          EXTZV   S^#VA$V_VPN,-           ; Get virtual page number
            52   52   15      04DC  2030                  S^#VA$S_VPN,R2,R2       ;
        50   00000000'GF  D0  04DF  2031          MOVL    G^MMG$GL_SPTBASE,R0     ; Get the base address of the SPT
             78 A5    6042  DE  04E6  2032          MOVAL   (R0)[R2],UCB$L_SVAPTE(R5) ; Set address of the SPT entry
                         04EB  2033 ;
                         04EB  2034 ; The following instruction also sets the data path number to the Direct
                         04EB  2035 ; Data Path.
                         04EB  2036 ;
                         04EB  2037          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                         04EB  2038          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                38 A4    D0  04EB  2039          MOVL    CDB_L_XMTMAP(R4),-      ; Assume we use preallocated map
                34 A6        04EE  2040                  CRB$L_INTD+VEC$W_MAPREG(R6) ; register.
                    57   D5  04F0  2041          TSTL    R7                      ; Is mapping slot the preallocated one?
                    1B   13  04F2  2042          BEQL    50$                     ; Br if yes - all set
                             04F4  2043                                          ; Else, allocate the map registers
                         04F4  2044 ;
                         04F4  2045 ; Allocate UNIBUS map registers
                         04F4  2046 ;
                         04F4  2047 ;;&&     CLRB    CRB$L_INTD+VEC$B_DATAPATH(R6) ; Reset data path usage
          00000000'GF    16  04F4  2048          JSB     G^IOC$ALOUBAMAP         ; Allocate UNIBUS map registers
```

```
                0C 50   E8  04FA  2049          BLBS    R0,40$                          ; Br if one available
          00DC C4   63   0E  04FD  2050          INSQUE  (R3),CDB_Q_XMTREQ(R4)           ; Re-insert CXB on request queue
                         0502  2051  20$:       POPQ    R6                              ; Restore R6,R7
                50   01   9A  0505  2052          MOVZBL  S^#SS$_NORMAL,R0                ; Good return
                         05  0508  2053  30$:     RSB                                     ; Return to caller
                             0509  2054  ;
                             0509  2055  ; Save the map information and map the buffer.
                             0509  2056  ;
                             0509  2057  40$:     ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                             0509  2058          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                   34 A6     D0  0509  2059          MOVL    CRB$L_INTD+VEC$W_MAPREG(R6),-   ; Save mapping info
                38 A447          050C  2060                  CDB_L_XMTMAP(R4)[R7]            ; in CDB
                             050F  2061  50$:     SETBIT  R7,CDB_B_XMTMAP(R4)             ; Set mapping slot in use flag
                 22 A3   57  90  0514  2062          MOVB    R7,CXB$B_XQ_SLOT(R3)            ; Save mapping slot number used
              00000000'GF   16  0518  2063          JSB     G^IOC$LOADUBAMAPA               ; Load map registers
                             051E  2064  ;
                             051E  2065  ; Find next ring entry and insert data
                             051E  2066  ;
                52   18 A4   9A  051E  2067          MOVZBL  CDB_B_NEXTXMT(R4),R2            ; Get next ring entry
                   18 A4   96  0522  2068          INCB    CDB_B_NEXTXMT(R4)               ; Bump ring pointer
                   FC 8F   8A  0525  2069          BICB    #^C<MAX_C_XMT-1>,-              ; Modulo xmit ring size
                   18 A4       0528  2070                  CDB_B_NEXTXMT(R4)               ;
                 23 A3   52  90  052A  2071          MOVB    R2,CXB$B_XQ_RING(R3)            ; Save ring entry number
             52   009C C442  D0  052E  2072          MOVL    CDB_L_XRINGVA(R4)[R2],R2        ; Get ring entry virtual address
                      62  B4  0534  2073          CLRW    XMT_W_FLAG(R2)                  ; Zero the FLAG word
                   8000 8F   B0  0536  2074          MOVW    #XMT_STS_M_LAST,-               ; Init STATUS word
                     08 A2       053A  2075                  XMT_Q_STS(R2)                   ;
                50   1A A3   3C  053C  2076          MOVZWL  CXB$W_BCNT(R3),R0               ; Get BYTE count
              0040 8F    50   B1  0540  2077          CMPW    R0,#64                          ; Is packet at least minimum size?
                         07   1E  0545  2078          BGEQU   60$                             ; Br if yes, okay
           50   00000040 8F   D0  0547  2079          MOVL    #64,R0                          ; Else set to minimum
                         50   D6  054E  2080  60$:     INCL    R0                              ; Round up by one
                50   50   FF 8F   78  0550  2081          ASHL    #-1,R0,R0                       ; Convert to WORD count
                     06 A2   50   AE  0555  2082          MNEGW   R0,XMT_W_LEN(R2)                ; Store message length (2's complement)
                   04 A2   7C   A5  B0  0559  2083          MOVW    UCB$W_BOFF(R5),XMT_W_ADDR(R2) ; Move byte offset - BA0-BA8
                       34 A6   F0  055E  2084          INSV    CRB$L_INTD+VEC$W_MAPREG(R6),- ; Insert BA9-BA15
                04 A2   07      09  0561  2085                  #9,#7,XMT_W_ADDR(R2)          ;&
           50   34 A6   06      07  EF  0565  2086          EXTZV   #7,#6,CRB$L_INTD+VEC$W_MAPREG(R6),R0 ; Get BA16-BA21
                     02 A2   50   B0  056B  2087          MOVW    R0,XMT_W_ADDRHI(R2)             ; Insert BA16-BA21 & zero descriptor bits
                       56   52   D0  056F  2088          MOVL    R2,R6                           ; Save xmit ring entry address
                                    0572  2089          ;
                                    0572  2090          ; Descriptor bit settings are calculated as follows: (Even, Odd)
                                    0572  2091          ;
                                    0572  2092          ;       BEG     LEN     END
                                    0572  2093          ;       ---     ---     ---
                                    0572  2094          ;        E       E       E
                                    0572  2095          ;        E       O       O
                                    0572  2096          ;        O       E       O
                                    0572  2097          ;        O       O       E
                                    0572  2098          ;
                50    A000 8F   B0  0572  2099          MOVW    #XMT_DSC_M_VALID!-              ; Build descriptor flag
                                    0577  2100                  XMT_DSC_M_EOM,R0
                     11 7C A5   E9  0577  2101          BLBC    UCB$W_BOFF(R5),70$              ; Br if even byte boundary at BEGinning
                                    057B  2102          ;
                                    057B  2103          ; Beginning buffer address is Odd
                                    057B  2104          ;
                                    057B  2105          SETBIT  #XMT_DSC_V_BEGODD,R0            ; Else, beginning is Odd
```

```
                11 1A A3   E8  057F  2106          BLBS    CXB$W_BCNT(R3),80$      ; Br if odd LENgth, end is even
                           0583  2107          SETBIT  #XMT_DSC_V_ENDODD,R0    ; Else, ending is Odd
                   06 A6   B7  0587  2108          DECW    XMT_Q_LEN(R6)           ; Increase length by 1 (complemented)
                      08   11  058A  2109          BRB     80$                     ; Continue
                           058C  2110          ;
                           058C  2111          ; Beginning buffer address is Even
                           058C  2112          ;
                04 1A A3   E9  058C  2113 70$:    BLBC    CXB$W_BCNT(R3),80$      ; Br if even LENgth
                           0590  2114          SETBIT  #XMT_DSC_V_ENDODD,R0    ; Else, ending is Odd
                      01   91  0594  2115 80$:    CMPB    #XQ_FC_V_XMIT,-         ; Is this a normal XMIT?
                   20 A3       0596  2116                  CXB$B_XQ_FUNC(R3)       ;
                      04   13  0598  2117          BEQL    85$                     ; Br if yes
                           059A  2118          SETBIT  #XMT_DSC_V_SETUP,R0     ; Else, indicate SETUP function
                   0A A6   B4  059E  2119 85$:    CLRW    XMT_Q_TDR(R6)           ; Clear the TDR cell
             02 A6    50   A8  05A1  2120          BISW    R0,XMT_W_ADDRHI(R6)     ; Set descriptor bits
                01 0F C4   96  05A5  2121          INCB    CDB_B_XMTCNT(R4)        ; Tally one more xmit in progress
                           05A9  2122          ;
                           05A9  2123          ; Request and load the port
                           05A9  2124          ;
                           05A9  2125          POPQ    R6                      ; Restore R6,R7
                           05AC  2126          DSBINT  UCB$B_DIPL(R5)          ; Sync access to device
          00E8 D4    63   0E  05B3  2127          INSQUE  (R3),@CDB_Q_INPUT+4(R4) ; Insert at end of input queue
                      03   12  05B8  2128          BNEQ    90$                     ; Br if not first entry on queue
                    1041   30  05BA  2129          BSBW    LOAD_PORT               ; Load port
                           05BD  2130 90$:    ENBINT                          ; Restore IPL
                    FED0   31  05C0  2131          BRW     XMT_ALT_START           ; Try for more requests
                           05C3  2132
                           05C3  2133 XMT_UV1:                                ; Transmit operation on u-VAX I
                           05C3  2134          ASSUME  MAX_C_XMTUV1 LE 8
                           05C3  2135          ASSUME  MAX_C_XMTUV1 LT MAX_C_XMT ; Must not use all rings
57   1B A4    01    00   EB  05C3  2136          FFC     #0,#MAX_C_XMTUV1,CDB_B_XMTMAP(R4),R7 ; Find a free transmit slot
                      7F   13  05C9  2137          BEQL    120$                    ; Br if none free
                           05CB  2138          ;
                           05CB  2139          ; Move CXB data to contiguous buffer.
                           05CB  2140          ;
  53    00DC D4    0F   05CB  2141          REMQUE  @CDB_Q_XMTREQ(R4),R3    ; Get oldest xmit request
                      78   1D  05D0  2142          BVS     120$                    ; Br if none
     52   18 A3    3C   05D2  2143          MOVZWL  CXB$W_BOFF(R3),R2       ; Get offset to start of data
          52   53   C0   05D6  2144          ADDL    R3,R2                   ; Compute system buffer virtual address
51    00D8 C447    D0   05D9  2145          MOVL    CDB_L_XMT_VA(R4)[R7],R1 ; Get contiguous buffer's VA
                      38   BB  05DF  2146          PUSHR   #^M<R3,R4,R5>           ; Save registers
61   62    1A A3    28   05E1  2147          MOVC3   CXB$W_BCNT(R3),(R2),(R1) ; Copy the data
                      38   BA  05E6  2148          POPR    #^M<R3,R4,R5>           ; Restore registers
                           05E8  2149          ;
                           05E8  2150          ; Find next ring entry and insert data
                           05E8  2151          ;
     52   18 A4    9A   05E8  2152          MOVZBL  CDB_B_NEXTXMT(R4),R2    ; Get next ring entry
          18 A4    96   05EC  2153          INCB    CDB_B_NEXTXMT(R4)       ; Bump ring pointer
          FC 8F    8A   05EF  2154          BICB    #^C<MAX_C_XMT-1>,-      ; Modulo xmit ring size
          18 A4         05F2  2155                  CDB_B_NEXTXMT(R4)       ;
     23 A3    52   90   05F4  2156          MOVB    R2,CXB$B_XQ_RING(R3)    ; Save ring entry number
     22 A3    57   90   05F8  2157          MOVB    R7,CXB$B_XQ_SLOT(R3)    ; Save mapping slot number used
                      05FC  2158          SETBIT  R7,CDB_B_XMTMAP(R4)     ; Set mapping slot in use flag
52    009C C442    D0   0601  2159          MOVL    CDB_L_XRINGVA(R4)[R2],R2; Get ring entry virtual address
                   62   B4  0607  2160          CLRW    XMT_W_FLAG(R2)          ; Zero the FLAG word
             8000 8F    B0   0609  2161          MOVW    #XMT_STS_M_LAST,-       ; Init STATUS word
                08 A2         060D  2162                  XMT_Q_STS(R2)           ;
```

```
                50   1A A3   3C  060F  2163          MOVZWL  CXB$W_BCNT(R3),R0       ; Get BYTE count
            0040 8F     50   B1  0613  2164          CMPW    R0,#64                  ; Is packet at least minimum size?
                        07   1E  0618  2165          BGEQU   110$                    ; Br if yes, okay
         50   00000040 8F   D0  061A  2166          MOVL    #64,R0                  ; Else set to minimum
                        50   D6  0621  2167 110$:     INCL    R0                      ; Round up by one
          50    50     FF 8F   78  0623  2168          ASHL    #-1,R0,R0               ; Convert to WORD count
                06 A2    50   AE  0628  2169          MNEGW   R0,XMT_W_LEN(R2)        ; Store message length (2's complement)
           50    00C0 C447   D0  062C  2170          MOVL    CDB_L_XMT_PA(R4)[R7],R0 ; Get contiguous buffer's PA
                04 A2    50   B0  0632  2171          MOVW    R0,XMT_W_ADDR(R2)       ; Move buffer address - BA00-BA08
          50    50     F0 8F   78  0636  2172          ASHL    #-16,R0,R0              ; Shift down hi order address lines
                02 A2    50   B0  063B  2173          MOVW    R0,XMT_W_ADDRHI(R2)     ; Insert BA16-BA21 & zero descriptor bits
                   56    52   D0  063F  2174          MOVL    R2,R6                   ; Save xmit ring entry address
                                0642  2175          ;
                                0642  2176          ; Descriptor bit settings are calculated as follows: (Even, Odd)
                                0642  2177          ;
                                0642  2178          ;       BEG     LEN     END
                                0642  2179          ;       ---     ---     ---
                                0642  2180          ;        E       E       E
                                0642  2181          ;        E       O       O
                                0642  2182          ;        O       E       O
                                0642  2183          ;        O       O       E
                                0642  2184          ;
              50    A000 8F     B0  0642  2185          MOVW    #XMT_DSC_M_VALID!-      ; Build descriptor flag
                                0647  2186                    XMT_DSC_M_EOM,R0
                    FF42        31  0647  2187          BRW     70$                     ; Continue - always even start buffers!
                                064A  2188
                                064A  2189 120$:     POPQ    R6                      ; Restore R6,R7
                    50    01    9A  064D  2190          MOVZBL  S^#SS$_NORMAL,R0        ; Good return
                                05  0650  2191          RSB                             ; Return to caller
                                0651  2192
                                0651  2193          .DSABL  LSB
                                0651  2194
```

```
                     0651  2196          .SBTTL  RCV_FDT - RECEIVE I/O OPERATION FDT ROUTINE
                     0651  2197 ;++
                     0651  2198 ; RCV_FDT - RECEIVE I/O OPERATION FDT ROUTINE
                     0651  2199 ;
                     0651  2200 ; Functional description:
                     0651  2201 ;
                     0651  2202 ; The specified buffer is checked for accessibility. The buffer address and count
                     0651  2203 ; are saved in the packet. Then IPL is set to device fork IPL and if a message is
                     0651  2204 ; available the operation is completed; otherwise, the packet is queued onto
                     0651  2205 ; the waiting receive list.
                     0651  2206 ;
                     0651  2207 ; The QIO parameters for WRITES are:
                     0651  2208 ;       P1 = Address of the data buffer
                     0651  2209 ;       P2 = Size of the data buffer
                     0651  2210 ;       P5 = Optional address of the buffer to receive the source address
                     0651  2211 ;
                     0651  2212 ; Inputs:
                     0651  2213 ;
                     0651  2214 ;       R3 = IRP address
                     0651  2215 ;       R4 = PCB address
                     0651  2216 ;       R5 = UCB address
                     0651  2217 ;       R6 = CCB address
                     0651  2218 ;       R7 = Function code
                     0651  2219 ;       AP = Address of the first operation specific qio parameter
                     0651  2220 ;
                     0651  2221 ;       IPL = ASTDEL
                     0651  2222 ;
                     0651  2223 ; Outputs:
                     0651  2224 ;
                     0651  2225 ;       R0 = Status of the receive qio operation
                     0651  2226 ;       R3 = IRP address
                     0651  2227 ;       R5 = UCB ADDRESS
                     0651  2228 ;
                     0651  2229 ;       R1,R2 are destroyed.
                     0651  2230 ;--
                     0651  2231
                     0651  2232 RCV_FDT::                                   ; Read operation FDT
                     0651  2233 ;
                     0651  2234 ;  Check the request params
                     0651  2235 ;
          30 A3   B4 0651  2236          CLRW    IRP$W_BOFF(R3)              ; Set no quota to here
        50    0C   9A 0654  2237          MOVZBL  S^#SS$_ACCVIO,R0            ; Assume access violation
     57   10 AC   D0 0657  2238          MOVL    P5(AP),R7                   ; Get address for source address
             33   13 065B  2239          BEQL    10$                         ; Br if none
                     065D  2240          IFNOWRT #RHDR_C_DATA,(R7),20$       ; Check for write access to buffer
             07   E0 0663  2241          BBS     #IRP$V_DIAGBUF,-            ; Br if diagnostic buffer given
       28 2A A3         0665  2242                  IRP$W_STS(R3),10$           ;       ...
        51   1A   9A 0668  2243          MOVZBL  #RHDR_C_LENGTH,R1           ; Get size of header buffer
             53   DD 066B  2244          PUSHL   R3                          ; Save IRP address
 00000000'GF   16   DD 066D  2245          JSB     G^EXE$ALLOCBUF              ; Allocate header buffer
             53 8ED0 0673  2246          POPL    R3                          ; Restore IRP address
        40   50   E9 0676  2247          BLBC    R0,20$                      ; Br if allocation failure
     4C A3   52   D0 0679  2248          MOVL    R2,IRP$L_DIAGBUF(R3)        ; Save buffer address
2A A3   0080 8F   A8 067D  2249          BISW    #IRP$M_DIAGBUF,IRP$W_STS(R3) ; Indicate diag buffer present
                     0683  2250          ASSUME  RHDR_L_DATA EQ 0
     82   0C A2   9E 0683  2251          MOVAB   RHDR_T_DATA(R2),(R2)+       ; Set address of start of data
                     0687  2252          ASSUME  RHDR_L_BUFFER EQ RHDR_L_DATA+4
```

```
                82    57    D0  0687  2253          MOVL    R7,(R2)+                        ; Set user buffer address
                                0068A 2254          ASSUME  RHDR_W_SIZE EQ RHDR_L_BUFFER+4
                82    51    B0  0068A 2255          MOVW    R1,(R2)+                        ; Save size of allocation
                                0068D 2256          ASSUME  RHDR_B_TYPE EQ RHDR_W_SIZE+2
                82    13    90  0068D 2257          MOVB    #DYN$C_BUFIO,(R2)+              ; Set structure type
                50    14    9A  0690  2258 10$:     MOVZBL  S^#SS$_BADPARAM,R0              ; Assume illegal size
          51    04  AC    3C  0693  2259          MOVZWL  P2(AP),R1                       ; Get size
                                0697  2260
                                0697  2261
                                0697  2262
                20    13    0697  2263          BEQL    20$                             ; Br if zero - illegal
                50    6C    D0  0699  2264          MOVL    P1(AP),R0                       ; Get user buffer address
          3C    A3    50    D0  069C  2265          MOVL    R0,IRP$L_XQ_DATBUF(R3)          ; Save user VA for completion
    00000000'GF       16    06A0  2266          JSB     G^EXE$READCHK                   ; Check the buffer
                                06A6  2267                                              ; (No return on NO ACCESS)
          2A    A3    20    A8  06A6  2268          BISW    #IRP$M_CHAINED,IRP$W_STS(R3) ; Allow data chaining
                                06AA  2269          SETIPL  UCB$B_FIPL(R5)                  ; Raise IPL to lock data base
                12    10    06AE  2270          BSBB    RCV_START                       ; Process the request
                06    50    E9  06B0  2271          BLBC    R0,20$                          ; Br if error
    00000000'GF       17    06B3  2272          JMP     G^EXE$QIORETURN                 ; Else, take normal return
                                06B9  2273
                FC8A    31    06B9  2274 20$:     BRW     ABORTIO                         ; Abort the request
```

```
                     06BC  2276          .SBTTL  RCV_START - START RECEIVE I/O OPERATION
                     06BC  2277 ;++
                     06BC  2278 ; RCV_START - START RECEIVE I/O OPERATION
                     06BC  2279 ;
                     06BC  2280 ; Functional description:
                     06BC  2281 ;
                     06BC  2282 ;  Check for device active.  Receives cannot be queued to the UCB receive
                     06BC  2283 ;  queue unless the UCB has been initialized via routine START.   They
                     06BC  2284 ;  cannot be put in the IRP queue since this could result in non-sequential
                     06BC  2285 ;  receive processing due to the existence of the separate receive queue.
                     06BC  2286 ;
                     06BC  2287 ; Inputs:
                     06BC  2288 ;
                     06BC  2289 ;       R3 = IRP address
                     06BC  2290 ;       R5 = UCB address
                     06BC  2291 ;
                     06BC  2292 ;       IPL = FIPL
                     06BC  2293 ;
                     06BC  2294 ; Outputs:
                     06BC  2295 ;
                     06BC  2296 ;       R0 = Status of receive request
                     06BC  2297 ;       R4 = CDB address
                     06BC  2298 ;
                     06BC  2299 ;       R1,R2 are destroyed.
                     06BC  2300 ;
                     06BC  2301 ;--
                     06BC  2302 INACT_ERROR:
          50   20D4 8F   3C  06BC  2303          MOVZWL  #SS$_DEVINACT,R0        ; Setup return status
                       05  06C1  2304          RSB                             ; Return to caller
                     06C2  2305
                     06C2  2306          .ENABL  LSB
                     06C2  2307 RCV_START::                                     ; Start receive I/O operation
                04   E1  06C2  2308          BBC     #UCB$V_XQ_RUN,-           ; Br if UCB is not in RUN mode
             F5 68   A5       06C4  2309                  UCB$W_DEVSTS(R5),INACT_ERROR
                     06C7  2310
          54   24   A5  D0  06C7  2311          MOVL    UCB$L_CRB(R5),R4          ; Get CRB address
          54   10   A4  D0  06CB  2312          MOVL    CRB$L_AUXSTRUC(R4),R4     ; Get CDB address
                39   13  06CF  2313          BEQL    40$                       ; Br if none
                01   E1  06D1  2314          BBC     #CDB_STS_V_RUN,-          ; Br if QNA not running
           E5 024A   C4       06D3  2315                  CDB_B_STS(R4),INACT_ERROR
       51   00A0   C5  9E  06D7  2316          MOVAB   UCB$Q_XQ_RCVMSG(R5),R1  ; Get address of UCB received messages
                     06DC  2317 ;
                     06DC  2318 ; If running in SHARED mode, then use the listheads in the SHR_
                     06DC  2319 ; data structure.
                     06DC  2320 ;
                03   E1  06DC  2321          BBC     #UCB$V_XQ_SHARE,-         ; Br if UCB is NOT SHARED
             09 68   A5       06DE  2322                   UCB$W_DEVSTS(R5),5$   ;
                     06E1  2323 ;
                     06E1  2324 ; Try to find a match on PID/CHAN
                     06E1  2325 ;
                0027 30  06E1  2326          BSBW    MATCH_SHR                 ; Try to find shared user
                  D6 12  06E4  2327          BNEQ    INACT_ERROR               ; Br if none - inactive user
          51   18   A1  9E  06E6  2328          MOVAB   SHR_Q_RCVMSG(R1),R1     ; Get address of received messages
                     06EA  2329 ;
                     06EA  2330 ; Check to see if message is available
                     06EA  2331 ;
          52   91   0F  06EA  2332 5$:       REMQUE  @(R1)+,R2                 ; Dequeue a received message
```

```
                    06ED  2333                                    ; ..bump pointer to end of list pointer
                 05 1D  06ED  2334          BVS     10$          ; Br if none
                    06EF  2335 ;
                    06EF  2336 ; Complete receive with available message
                    06EF  2337 ;
               1421 30  06EF  2338          BSBW    FINISH_RCV_IO ; Complete the receive
                 13 11  06F2  2339          BRB     30$          ; And exit
                    06F4  2340 ;
                    06F4  2341 ; Queue IRP for future message arivial unless IO$M_NOW specified
                    06F4  2342 ;
 0A 20 A3       06 E1  06F4  2343 10$:      BBC     #IO$V_NOW,IRP$W_FUNC(R3),20$ ; Br if not READ NOW
 50     0870 8F    3C  06F9  2344          MOVZWL  #SS$_ENDOFFILE,R0 ; Set no message status
           14DB 30  06FE  2345          BSBW    IO_DONE      ; Complete the I/O
             04 11  0701  2346          BRB     30$          ; And exit
                    0703  2347 ;
                    0703  2348 ; Queue the IRP to UCB receive wait queue
                    0703  2349 ;
                    0703  2350          ASSUME  UCB$Q_XQ_RCVREQ EQ UCB$Q_XQ_RCVMSG+8
                    0703  2351          ASSUME  SHR_Q_RCVREQ EQ SHR_Q_RCVMSG+8
 08 B1          63 0E  0703  2352 20$:      INSQUE  (R3),@8(R1)  ; Put packet on waiting list
    50          01 9A  0707  2353 30$:      MOVZBL  S^#SS$_NORMAL,R0 ; Set QIO status return
                05 05  070A  2354 40$:      RSB                  ; Return to caller
                    070B  2355          .DSABL  LSB
```

```
                     070B  2357          .SBTTL   SUBROUTINES TO FIND SHR DATA STRUCTURE
                     070B  2358 ;+
                     070B  2359 ; Subroutine to find SHR data structure for user
                     070B  2360 ;
                     070B  2361 ; Inputs:
                     070B  2362 ;       R3 = Address of IRP
                     070B  2363 ;       R5 = UCB address
                     070B  2364 ;
                     070B  2365 ; Outputs:
                     070B  2366 ;       R1 = Address if SHR data structure if match
                     070B  2367 ;       R0 is destroyed.
                     070B  2368 ;       Z-Bit set then match.
                     070B  2369 ;       Z-Bit clear then no match.
                     070B  2370 ;-
                     070B  2371
                     070B  2372 MATCH_SHR:                                  ; Try to find shared user
51   00C4 C5   D0    070B  2373          MOVL    UCB$L_XQ_DEFUSR(R5),R1    ; Get address of default user
          04   13    0710  2374          BEQL    10$                       ; Br if no default user
          1C   10    0712  2375          BSBB    CHECK_SHR                 ; Check for match
          19   13    0714  2376          BEQL    40$                       ; Br if match
50   0098 C5   9E    0716  2377 10$:     MOVAB   UCB$Q_XQ_SHARE(R5),R0     ; Save address of listhead
     51   50   D0    071B  2378          MOVL    R0,R1                     ; Copy listhead address
                     071E  2379          ASSUME  SHR_L_QFL EQ 0
     51   61   D0    071E  2380 20$:     MOVL    (R1),R1                   ; Get next in list
     50   51   D1    0721  2381          CMPL    R1,R0                     ; Back to start of list?
          06   13    0724  2382          BEQL    30$                       ; Br if yes - no pid/chan match
          08   10    0726  2383          BSBB    CHECK_SHR                 ; Check for match
          F4   12    0728  2384          BNEQ    20$                       ; Br if none
          03   11    072A  2385          BRB     40$                       ; Return in success
     50   50   D0    072C  2386 30$:     MOVL    R0,R0                     ; Return match failure
          05   072F  2387 40$:     RSB
                     0730  2388
                     0730  2389 ;+
                     0730  2390 ; Subroutine to check if PID and SHR data base match up
                     0730  2391 ;
                     0730  2392 ; Inputs:
                     0730  2393 ;       R1 = Address of SHR
                     0730  2394 ;       R3 = Address of IRP
                     0730  2395 ;
                     0730  2396 ; Outputs:
                     0730  2397 ;       Z-Bit set then match.
                     0730  2398 ;       Z-Bit clear then no match.
                     0730  2399 ;-
                     0730  2400
                     0730  2401 CHECK_SHR:                                  ; Check for match with SHR data base
          0C A3   D5 0730  2402          TSTL    IRP$L_PID(R3)             ; Is this an Internal IRP user?
          09   19   0733  2403          BLSS    10$                       ; Br if yes, only one allowed per UCB
                     0735  2404 ;
                     0735  2405 ; Normal QIO user
                     0735  2406 ;
0C A1     0C A3   D1 0735  2407          CMPL    IRP$L_PID(R3),SHR_L_PID(R1) ; PIDs match?
          0F   12    073A  2408          BNEQ    30$                       ; Br if no - try for next
          08   11    073C  2409          BRB     20$                       ; Else, continue checks
                     073E  2410 ;
                     073E  2411 ; Internal IRP user
                     073E  2412 ;
0C A1     00B8 C5   D1 073E  2413 10$:     CMPL    UCB$L_XQ_PID(R5),SHR_L_PID(R1) ; Is this the Internal user?
```

```
                05   12  0744  2414              BNEQ    30$                             ; Br if not
          10 A1 28 A3   B1  0746  2415 20$:       CMPW    IRP$W_CHAN(R3),SHR_W_CHAN(R1) ; Channels match?
                05      074B  2416 30$:       RSB                                     ; Return to caller
                        074C  2417
```

```
                     074C  2419          .SBTTL  ALT_START - ALTERNATE START I/O ROUTINE
                     074C  2420 ;++
                     074C  2421 ; ALT_START - ALTERNATE STARTIO I/O ROUTINE
                     074C  2422 ;
                     074C  2423 ; Functional description:
                     074C  2424 ;
                     074C  2425 ; This routine is called by the Executive to pass an "internal" IRP
                     074C  2426 ; to the driver.  "Internal" IRP's are those not built via QIO.
                     074C  2427 ; These IRPs are used by higher level software used to request I/O and
                     074C  2428 ; should not be confused with the IRPs built and passed by the
                     074C  2429 ; Transport layer to NSP.  The action here is to setup the IRP fields
                     074C  2430 ; as if the packet had been processed by the FDT routines.
                     074C  2431 ;
                     074C  2432 ; Inputs:
                     074C  2433 ;
                     074C  2434 ;       R3 = IRP address
                     074C  2435 ;       R5 = UCB address
                     074C  2436 ;
                     074C  2437 ;       All pertinent fields of the IRP are assumed to be valid.
                     074C  2438 ;
                     074C  2439 ;       IPL = FIPL
                     074C  2440 ;
                     074C  2441 ; Implicit inputs:
                     074C  2442 ;
                     074C  2443 ;       IRP$L_SVAPTE(R3) = System VIRTUAL address (not physical PTE address)
                     074C  2444 ;
                     074C  2445 ; Outputs:
                     074C  2446 ;
                     074C  2447 ;       R0-R5 may be garbage
                     074C  2448 ;--
                     074C  2449
                     074C  2450 ALT_START::                                 ; Accept an "internal" IRP
          01    E0   074C  2451          BBS     #IRP$V_FUNC,-              ; If BS then read function
      2D 2A A3        074E  2452                  IRP$W_STS(R3),10$
       00CA C5    B0  0751  2453          MOVW    UCB$W_XQ_PROTYP(R5),-      ; MUST be a non-promiscous user
          3A A3        0755  2454                  IRP$W_XQ_PROTYP(R3)        ;
    52    2C A3    D0  0757  2455          MOVL    IRP$L_SVAPTE(R3),R2        ; Get address of start of data
51  62    0E   C3    075B  2456          SUBL3   #XQ_C_HEADER,(R2),R1       ; Get the xmit buffer address
      51    52   C2    075F  2457          SUBL    R2,R1                      ; Form offset to start of data
   18 A2    51   B0    0762  2458          MOVW    R1,CXB$W_BOFF(R2)          ; Store offset in CXB
   0A A2    1B   90    0766  2459          MOVB    #DYN$C_CXB,CXB$B_TYPE(R2)  ; Set structure type to CXB
       00D8 C5   91    076A  2460          CMPB    UCB$B_XQ_PRO(R5),-         ;% Point-to-point mode?
                 00    076E  2461                  #NMA$C_LINPR_POI
                 03    12    076F  2462          BNEQ    5$                 ;% If so,
          40 A3    7C    0771  2463          CLRQ    IRP$Q_STATION(R3)          ;% Pick up destination from SHR block
                 53    DD    0774  2464 5$:      PUSHL   R3                 ; Save IRP address
            FC16    30    0776  2465          BSBW    XMT_START              ; Start transmit operation
                 53 8ED0    0779  2466          POPL    R3                 ; Restore IRP address
                 17    11    077C  2467          BRB     30$                ; Continue
                       077E  2468
    52    2C A3    D0  077E  2469 10$:     MOVL    IRP$L_SVAPTE(R3),R2        ; Get address of input buffer
                 0E    13    0782  2470          BEQL    20$                ; Br if none
    54    24 A5    D0  0784  2471          MOVL    UCB$L_CRB(R5),R4           ; Get CRB address
    54    10 A4    D0  0788  2472          MOVL    CRB$L_AUXSTRUC(R4),R4      ; Get CDB address
          2C A3    D4  078C  2473          CLRL    IRP$L_SVAPTE(R3)           ; Make sure SVAPTE is cleared
                       078F  2474 ;
                       078F  2475 ; The driver must be prepared to process chained buffers returned from
```

```
                     078F   2476 ; the higher levels.
                     078F   2477 ;
                     078F   2478 ;&&15$: PUSHL   CXB$L_LINK(R2)          ; Save address of next in link
                     078F   2479 ;&&      CLRL    CXB$L_LINK(R2)          ; Clear the link cell
 0CAE    30          078F   2480          BSBW    ADDRCVLIST              ; Else, add it to the receive list
                     0792   2481 ;&&      POPL    R2                      ; Get back address of next in chain
                     0792   2482 ;&&      BNEQ    15$                     ; Br if more to return
                     0792   2483
 FF2D    30          0792   2484 20$:     BSBW    RCV_START               ; Start receive operation
01 50    E9          0795   2485 30$:     BLBC    R0,40$                  ; Br if error
         05          0798   2486          RSB                             ; Return to caller
                     0799   2487
 1440    31          0799   2488 40$:     BRW     IO_DONE                 ; Post the I/O request in error
```

```
079C  2490          .SBTTL  SETMODE_FDT - SET MODE I/O OPERATION FDT DISPATCH ROUTINE
079C  2491 ;++
079C  2492 ; SETMODE_FDT - SET MODE FDT PROCESSING
079C  2493 ;
079C  2494 ; Functional description:
079C  2495 ;
079C  2496 ; This is the fdt routine for setmode functions.
079C  2497 ; There are three functions based on subfunction modifier bit.
079C  2498 ;
079C  2499 ; NOTE: That there is no difference on a request to shutdown a line or
079C  2500 ;       a circuit. However, a request to startup a circuit is ignored
079C  2501 ;       completely.
079C  2502 ;
079C  2503 ; The QIO parameters for SETMODE are:
079C  2504 ;
079C  2505 ;       P2 = Optional address of buffer descriptor for extended characteristics
079C  2506 ;
079C  2507 ;
079C  2508 ; The Subfunction modifiers are as follows:
079C  2509 ;
079C  2510 ; 1)    CHANGE MODE -- NO MODIFIER BIT.
079C  2511 ;       This function is done in the STARTIO routine. Control is passed to
079C  2512 ;       EXE$SETMODE to validate the new mode buffer and queue the packet.
079C  2513 ;
079C  2514 ; 2)    INITIALIZE THE UNIT -- IO$M_STARTUP SET.
079C  2515 ;       This function is done partially here and the remainder in STARTIO.
079C  2516 ;       The action here is to pick up the user buffered I/O quota.  The quota
079C  2517 ;       taken from the user is in IRP$W_BOFF. This value will be the IOSB+2 value
079C  2518 ;       at I/O done. The mailbox is enabled and a receive is started.
079C  2519 ;
079C  2520 ; 3)    SHUTDOWN UNIT -- IO$M_SHUTDOWN SET.
079C  2521 ;       This function shuts down the unit and optionally resets the mode.
079C  2522 ;       A CANCEL I/O is performed, all outstanding I/O is completed, the
079C  2523 ;       message blocks are all returned and the unit is left in an idle
079C  2524 ;       state. This function cannot be done here and the FDT processing is
079C  2525 ;       that of all SETMODE operations.
079C  2526 ;
079C  2527 ; 4)    ATTENTION AST -- IO$M_ATTNAST SET.
079C  2528 ;       This function sets up an AST to be delivered when a change of
079C  2529 ;       status occurs on the QNA.
079C  2530 ;
079C  2531 ;
079C  2532 ; Inputs:
079C  2533 ;
079C  2534 ;       R3 = IRP ADDESS
079C  2535 ;       R4 = PCB ADDRESS
079C  2536 ;       R5 = UCB ADDRESS
079C  2537 ;       R6 = CCB ADDRESS
079C  2538 ;       R7 = FUNCTION CODE
079C  2539 ;       AP = ADDRESS OF THE FIRST QIO PARAMETER
079C  2540 ;
079C  2541 ;       IPL = ASTDEL
079C  2542 ;
079C  2543 ; Outputs:
079C  2544 ;
079C  2545 ;       R3 = IRP ADDRESS
079C  2546 ;       R4 = PCB ADDRESS
```

```
                     079C  2547 ;       R5 = UCB ADDRESS
                     079C  2548 ;
                     079C  2549 ;       R0-R2,R6 are destroyed.
                     079C  2550 ;
                     079C  2551 ;--
                     079C  2552
                     079C  2553 SETMODE_FDT::                                ; SET MODE FDT processing
          2C A3   D4 079C  2554          CLRL    IRP$L_SVAPTE(R3)            ; Set no buffered packet
          38 A3   7C 079F  2555          CLRQ    IRP$L_MEDIA(R3)             ; Reset mode data area
        0094 C3   D4 07A2  2556          CLRL    IRP$L_XQ_SETUP(R3)          ; Indicate no SETUP buffer yet
50      0084 8F   3C 07A6  2557          MOVZWL  #SS$_DEVOFFLINE,R0          ; Assume unit if offline
            04    E0 07AB  2558          BBS     #UCB$V_ONLINE,-             ; Br if unit online
        03 64 A5       07AD  2559                  UCB$W_STS(R5),5$            ;
             FB93   31 07B0  2560          BRW     ABORTIO                     ; Else, abort the I/O request
                    07B3  2561
  57      20 A3   B0 07B3  2562 5$:      MOVW    IRP$W_FUNC(R3),R7           ; Get entire function code
  5C 57     08    E1 07B7  2563          BBC     #IO$V_ATTNAST,R7,30$        ; Br if not attention AST
                    07BB  2564 ;
                    07BB  2565 ; User is requesting an attention AST
                    07BB  2566 ;
57      00C0 C5   DE 07BB  2567          MOVAL   UCB$L_XQ_AST(R5),R7         ; Get address of AST list
 00000000'GF      16 07C0  2568          JSB     G^COM$SETATTNAST            ; Set up attention AST
                    07C6  2569
                    07C6  2570          ASSUME  UCB$V_XQ_INITED  EQ  0
        12 68 A5   E9 07C6  2571          BLBC    UCB$W_DEVSTS(R5),10$        ; Br if protocol not active
                    07CA  2572
51      00A0 C5   9E 07CA  2573          MOVAB   UCB$Q_XQ_RCVMSG(R5),R1      ; Check for empty receive list
        61    51  D1 07CF  2574          CMPL    R1,(R1)                     ; Empty?
              08  13 07D2  2575          BEQL    10$                         ; Br if YES, no need to inform user
              53  DD 07D4  2576          PUSHL   R3                          ; Save IRP address
            24B8  30 07D6  2577          BSBW    POKE_USER                   ; Inform user
              53 8ED0 07D9  2578          POPL    R3                          ; Restore IRP address
        50    01  9A 07DC  2579 10$:     MOVZBL  S^#SS$_NORMAL,R0            ; Set success
  51    44 A5   D0 07DF  2580          MOVL    UCB$L_DEVDEPEND(R5),R1      ; Get device dependent information
00000000'GF     17 07E3  2581 20$:     JMP     G^EXE$FINISHIO              ; Complete the I/O
                    07E9  2582 ;
                    07E9  2583 ; On a circuit request,
                    07E9  2584 ;       If this is a shutdown then perform a $CANCEL and clear the RUN flag.
                    07E9  2585 ;       If this is a startup then set the RUN flag.
                    07E9  2586 ;
14 57     07    E1 07E9  2587 25$:     BBC     #IO$V_SHUTDOWN,R7,27$       ; Br if not a shutdown request
                    07ED  2588          SETIPL  UCB$B_FIPL(R5)              ; Sync acces to UCB & CDB
                    07F1  2589          CLRBIT  #UCB$V_XQ_RUN,-             ; Clear the RUN flag
                    07F1  2590                  UCB$W_DEVSTS(R5)
52      28 A3   3C 07F6  2591          MOVZWL  IRP$W_CHAN(R3),R2           ; Get channel number
                    07FA  2592          ASSUME  CAN$C_CANCEL EQ 0
              58  D4 07FA  2593          CLRL    R8                          ; Set $CANCEL function
            1C73  30 07FC  2594          BSBW    CANCEL                      ; Perform a CANCEL
              DB  11 07FF  2595          BRB     10$                         ; Complete the request
                    0801  2596
D7 57     06    E1 0801  2597 27$:     BBC     #IO$V_STARTUP,R7,10$        ; Br if not a startup request
                    0805  2598          CLRBIT  #XM$V_ERR_START,UCB$L_DEVDEPEND(R5) ;% Clear start error flag
        0060 8F   A8 080A  2599          BISW    #UCB$M_XQ_START!UCB$M_XQ_STACK,- ;% Set start and stack states
            68 A5       080E  2600                  UCB$W_DEVSTS(R5)            ;%
                    0810  2601          SETBIT  #UCB$V_XQ_RUN,-             ; Set the RUN flag
                    0810  2602                  UCB$W_DEVSTS(R5)
                C5  11 0815  2603          BRB     10$                         ; Complete the request
```

```
                        0817  2604 ;
                        0817  2605 ; For everthing except Attention ASTs we must make sure CDB is present
                        0817  2606 ; and we must verify the P2 buffer.
                        0817  2607 ;
 51    24 A5    D0      0817  2608 30$:    MOVL    UCB$L_CRB(R5),R1          ; Get CRB address
         10 A1    D5     081B  2609         TSTL    CRB$L_AUXSTRUC(R1)        ; Is CDB there?
            0E    12     081E  2610         BNEQ    35$                       ; Br if yes
          175E    30     0820  2611         BSBW    ALLOC_CDB                 ; Else, allocate CDB
        08 50     E8     0823  2612         BLBS    R0,35$                    ; Br if successful
50    0124 8F     3C     0826  2613         MOVZWL  #SS$_INSFMEM,R0           ; Set error return
          FB18    31     082B  2614 33$:    BRW     ABORTIO                   ; Return error
                  082E  2615
          055F    30     082E  2616 35$:    BSBW    GET_CHAR_BUF              ; Get P2 characteristics
        F7 50     E9     0831  2617         BLBC    R0,33$                    ; Br if error - Abort I/O
52    F840 CF     9E     0834  2618         MOVAB   LINE_PARAM_WO,R2          ; Assume the line parameters
   05 57    09     E0     0839  2619         BBS     #IO$V_CTRL,R7,36$         ; Br if line request
52    F8BD CF     9E     083D  2620         MOVAB   CIRCUIT_PARAM,R2          ; Else, use the circuit parameters
          1F25    30     0842  2621 36$:    BSBW    VALIDATE_P2               ; Validate the P2 parameters
        9B 50     E9     0845  2622         BLBC    R0,20$                    ; Br if error
 9D 57    09     E1     0848  2623         BBC     #IO$V_CTRL,R7,25$         ; Br if not a LINE request
                  084C  2624         SETIPL  UCB$B_FIPL(R5)            ; Sync access to UCB's
          1EEB    30     0850  2625         BSBW    SAV_MULTI                 ; Save the multicast address list
       0090 C3    D4     0853  2626         CLRL    IRP$L_XQ_SHR(R3)                  ; Assume exclusive user
            03    E1     0857  2627         BBC     #UCB$V_XQ_SHARE,-         ; Br if not a SHARED user
     16 68 A5           0859  2628                 UCB$W_DEVSTS(R5),40$      ;
                  085C  2629 ;
                  085C  2630 ;       Allow the shared user to change the destination node with which
                  085C  2631 ;       it is communicating.
                  085C  2632 ;
          FEAC    30     085C  2633         BSBW    MATCH_SHR                 ; Else, try to find shared user
            11    12     085F  2634         BNEQ    40$                       ; Br if none found, skip it
0090 C3     51    D0     0861  2635         MOVL    R1,IRP$L_XQ_SHR(R3)       ; Save the SHR data structure address
         12 A1    D0     0866  2636         MOVL    SHR_G_DEST(R1),-          ; Save the current destination user
       00CC C5           0869  2637                 UCB$G_XQ_DES(R5)          ;  address in the UCB
         16 A1    B0     086C  2638         MOVW    SHR_G_DEST+4(R1),-        ;       ...
       00D0 C5           086F  2639                 UCB$G_XQ_DES+4(R5)        ;       ...
                  0872  2640 ;
                  0872  2641 ;       Now we will set the parameters given in the setmode request. But,
                  0872  2642 ;       first if the DEQNA controller is inited we will use the current
                  0872  2643 ;       hardware settings for the defaults.
                  0872  2644 ;
 51    24 A5    D0      0872  2645 40$:    MOVL    UCB$L_CRB(R5),R1          ; Get CRB address
 51    10 A1    D0      0876  2646         MOVL    CRB$L_AUXSTRUC(R1),R1     ; Get CDB address
                  087A  2647         ASSUME  CDB_STS_V_INITED EQ 0
 13 024A C1     E9      087A  2648         BLBC    CDB_B_STS(R1),45$         ; Br if controller not enabled, use
                  087F  2649                                                   ;  the fixed defaults
51    024D C1     9E     087F  2650         MOVAB   CDB_B_SETPRM(R1),R1       ; Get address of setable parameters
52    00DD C5     9E     0884  2651         MOVAB   UCB$B_XQ_CDBPRM(R5),R2    ; Get address of UCB parameters
      50    01    9A     0889  2652         MOVZBL  #UCB$C_XQ_CDBPRM,R0       ; Get size of parameters to move
      82    81    90     088C  2653 42$:    MOVB    (R1)+,(R2)+               ; Store CDB parameters into UCB
         FA 50    F5     088F  2654         SOBGTR  R0,42$                    ; Loop if more
52    F7E2 CF     9E     0892  2655 45$:    MOVAB   LINE_PARAM_WO,R2          ; Get address of verification table
          200E    30     0897  2656         BSBW    CHANGE_PARAM              ; Change the parameters
51    0090 C3     D0     089A  2657         MOVL    IRP$L_XQ_SHR(R3),R1       ; Get the SHR structure address
            0C    13     089F  2658         BEQL    50$                       ; Br if not present, skip it
       00CC C5    D0     08A1  2659         MOVL    UCB$G_XQ_DES(R5),-        ; Else, reset the destination user
         12 A1           08A5  2660                 SHR_G_DEST(R1)            ;  address into the SHR structure
```

```
                00D0 C5    B0  08A7  2661          MOVW    UCB$G_XQ_DES+4(R5),-    ;       ...
                  16 A1        08AB  2662                  SHR_G_DEST+4(R1)        ;       ...
                               08AD  2663
               0E 57   07    E1  08AD  2664  50$:   BBC     #IO$V_SHUTDOWN,R7,60$   ; Br if not shutdown request
                               08B1  2665  ;
                               08B1  2666  ; Shutdown protocol request
                               08B1  2667  ;
                               08B1  2668          ASSUME  UCB$V_XQ_INITED  EQ  0
                  03 68 A5     E8  08B1  2669          BLBS    UCB$W_DEVSTS(R5),55$    ; Br if still inited
                    FF24       31  08B5  2670          BRW     10$                     ; Else, complete I/O request now
               21 A3   03      90  08B8  2671  55$:   MOVB    S^#XQ_FC_V_STOP,IRP$B_XQ_FUNC(R3) ; Set internal function code
                    0404       31  08BC  2672          BRW     QUEPKT                  ; Queue request to QNA
                               08BF  2673
               03 57   06      E0  08BF  2674  60$:   BBS     #IO$V_STARTUP,R7,80$    ; Br if startup function
                    0108       31  08C3  2675  70$:   BRW     180$                    ; Else, must be change mode
                               08C6  2676  ;
                               08C6  2677  ; Startup protocol request
                               08C6  2678  ;
               21 A3   00      90  08C6  2679  80$:   MOVB    #XQ_FC_V_INIT,IRP$B_XQ_FUNC(R3) ; Insert internal function code
                               08CA  2680  ;
                               08CA  2681  ;       If the UCB is already initialized for SHARED use, then we will
                               08CA  2682  ;       check to make sure that the SHR struture exists. If it does then
                               08CA  2683  ;       the share structure must be active, by definition.
                               08CA  2684  ;
                  0090 C3      D5  08CA  2685          TSTL    IRP$L_XQ_SHR(R3)                ; Was the SHR structure present
                       F3      12  08CE  2686          BNEQ    70$                     ; Br if yes, already started
                       03      E0  08D0  2687          BBS     #UCB$V_XQ_SHARE,-       ; Br if SHARED UCB, ingore status
                  64 68 A5         08D2  2688                  UCB$W_DEVSTS(R5),125$   ; ..make user a shared user
                                   08D5  2689          ASSUME  UCB$V_XQ_INITED  EQ  0
                  EA 68 A5     E8  08D5  2690          BLBS    UCB$W_DEVSTS(R5),70$    ; Br if already started
                                   08D9  2691                                          ; ..check multicast address list
                       02      E1  08D9  2692          BBC     #UCB$V_XQ_PROTYP,-      ; Br if no protocol specified
                  46 68 A5         08DB  2693                  UCB$W_DEVSTS(R5),100$   ; ..error
                                   08DE  2694  ;
                                   08DE  2695  ; Check if protocol type is to be shared
                                   08DE  2696  ;
                       03      91  08DE  2697          CMPB    #NMA$C_ACC_EXC,-        ; Is this PROTOCOL TYPE for exclusive
                    00D4 C5        08E0  2698                   UCB$B_XQ_ACC(R5)       ;  use?
                       54      12  08E3  2699          BNEQ    125$                    ; Br if not
                                   08E5  2700  ;
                                   08E5  2701  ; Check protocol type for uniqueness
                                   08E5  2702  ;
                  51   00CA C5     3C  08E5  2703          MOVZWL  UCB$W_XQ_PROTYP(R5),R1  ; Get protocol type
                                   08EA  2704  ;
                                   08EA  2705  ; For a user wishing to run in promiscuous mode, the requirement is that
                                   08EA  2706  ; there be no other promiscuous users running. For a non-promiscuous user,
                                   08EA  2707  ; there must be no other users running with the same protocol type.
                                   08EA  2708  ;
                50   00002CC6'EF   9E  08EA  2709          MOVAB   MATCH_PROTYP,R0         ; Get address of Action routine
                                   08F1  2710                                          ;  assume non-PROMISCUOUS user
                                   08F1  2711
                                   08F1  2712          ASSUME  NMA$C_STATE_ON EQ 0
                                   08F1  2713          ASSUME  NMA$C_STATE_OFF EQ 1
                                   08F1  2714
                    1B 00DA C5     E8  08F1  2715          BLBS    UCB$B_XQ_PRM(R5),90$    ; Br if NOT a PROMISCUOUS user
                                   08F6  2716  ;
                                   08F6  2717  ; The promiscuous user must have PHY_IO privilege
```

```
                        08F6  2718 ;
50  00002CE8'EF   9E    08F6  2719          MOVAB   MATCH_PROMTYP,R0          ; Get address of Action routine
                        08FD  2720          IFPRIV  PHY_IO,90$               ; If user has privilege, then okay
          1E4E  30      0903  2721          BSBW    RES_MULTI                ; Else, Restore original multicast list
       50   24  3C      0906  2722          MOVZWL  #SS$_NOPRIV,R0           ; Return error - NOPRIV
    51   0B18 8F  3C      0909  2723          MOVZWL  #NMA$C_PCLI_PRM,R1       ; Return the bad parameter
          FED2  31      090E  2724          BRW     20$                      ; Finish the I/O request
                        0911  2725 90$:     PUSHQ   R4                       ; Save PCB, UCB addresses
      54   24 A5  D0      0914  2726          MOVL    UCB$L_CRB(R5),R4         ; Get CRB address
      54   10 A4  D0      0918  2727          MOVL    CRB$L_AUXSTRUC(R4),R4    ; Get CDB address
             60  16      091C  2728          JSB     (R0)                     ; Try to find exact match
                        091E  2729          POPQ    R4                       ; Restore PCB, UCB addresses
          22 50  E9      0921  2730          BLBC    R0,130$                  ; Br if none found - okay
                        0924  2731 ;
                        0924  2732 ;       Bad protocol type
                        0924  2733 ;
    51   0B0E 8F  3C      0924  2734 100$:    MOVZWL  #NMA$C_PCLI_PTY,R1       ; Return bad parameter code
             05  11      0929  2735          BRB     120$                     ; Finish error reporting
                        092B  2736 ;
                        092B  2737 ;       Bad quota calculated
                        092B  2738 ;
    51   0451 8F  3C      092B  2739 110$:    MOVZWL  #NMA$C_PCLI_BFN,R1       ; Return bad parameter code
       50   14  9A      0930  2740 120$:    MOVZBL  S^#SS$_BADPARAM,R0       ; Set error return
          1E1E  30      0933  2741 123$:    BSBW    RES_MULTI                ; Restore original multicast list
          FEAA  31      0936  2742          BRW     20$                      ; Finish the I/O request
                        0939  2743 ;
                        0939  2744 ; Shared protocol type - look for same protocol type in other UCB.
                        0939  2745 ;
          013C  30      0939  2746 125$:    BSBW    SHR_UCB                  ; Find other UCB in user or make this
                        093C  2747                                           ;  UCB shareable
       F4 50  E9      093C  2748          BLBC    R0,123$                  ; Br on error
                        093F  2749 ;
                        093F  2750 ; We must now check if UCB is already inited.
                        093F  2751 ;
                        093F  2752          ASSUME  UCB$V_XQ_INITED  EQ  0
    11 68 A5  E9      093F  2753          BLBC    UCB$W_DEVSTS(R5),135$    ; Br if this UCB is NOT inited
                        0943  2754                                           ; skip quota taking, already done
          008F  31      0943  2755          BRW     185$                     ; Else, compute multicast list
                        0946  2756
                        0946  2757 130$:    SETIPL  #IPL$_ASTDEL             ; Reset IPL to ASTDEL
                        0949  2758 ;
                        0949  2759 ; Take quota needed
                        0949  2760 ;
          009A  30      0949  2761          BSBW    TAKE_QUOTA               ; Take quota from user
        DC 50  E9      094C  2762          BLBC    R0,1T0$                  ; Br if error
00C8 C5   57  B0      094F  2763          MOVW    R7,UCB$W_XQ_QUOTA(R5)    ; Save quota in UCB
                        0954  2764 135$:    SETBIT  #UCB$V_XQ_INITED,-       ; Indicate unit is initialized
                        0954  2765                  UCB$W_DEVSTS(R5)         ;       ...
                        0959  2766          SETIPL  UCB$B_FIPL(R5)           ; Sync access to UCB and CDB
  0D 00DA C5  E8      095D  2767          BLBS    UCB$B_XQ_PRM(R5),140$    ; Br if NOT a PROMISCUOUS user
  54   24 A5  D0      0962  2768          MOVL    UCB$L_CRB(R5),R4         ; Get CRB address
  54   10 A4  D0      0966  2769          MOVL    CRB$L_AUXSTRUC(R4),R4    ; Get CDB address
0214 C4   55  D0      096A  2770          MOVL    R5,CDB_L_PRMUSER(R4)     ; Store promiscuous user's address
                        096F  2771 ;
                        096F  2772 ; Pre-allocate all needed receive buffers, if the CDB is not initialized yet!
                        096F  2773 ; The buffers are immediately deallocated, but this pre-allocation will allow
                        096F  2774 ; the pool to grow if necessary! This must be done here, before we run on the
```

```
                        096F  2775 ; interrupt stack.
                        096F  2776 ;
                        096F  2777 140$:    PUSHQ   R3                          ; Save R3, R4
51   00D6 C5    3C      0972  2778          MOVZWL  UCB$W_XQ_BSZ(R5),R1         ; Get size of receive buffer
54   00D2 C5    3C      0977  2779          MOVZWL  UCB$W_XQ_HBQ(R5),R4         ; Get device buffer quota
       54  51    C6      097C  2780          DIVL    R1,R4                       ; Compute number of buffers to allocate
           54    D6      097F  2781          INCL    R4                          ; Plus one extra
           7E    D4      0981  2782          CLRL    -(SP)                       ; End of list marker
 00000000'GF     16      0983  2783 145$:    JSB     G^EXE$ALONONPAGED           ; Allocate the memory
         0E 50   E9      0989  2784          BLBC    R0,150$                     ; Br on error
  08 A2    51    B0      098C  2785          MOVW    R1,IRP$W_SIZE(R2)           ; Save size of buffer
  0A A2    1B    90      0990  2786          MOVB    #DYN$C_CXB,IRP$B_TYPE(R2) ; Set structure type
       7E  52    D0      0994  2787          MOVL    R2,-(SP)                    ; Save buffer address
         E9 54   F5      0997  2788          SOBGTR  R4,145$                     ; Loop if more to allocate
                        099A  2789
       50   8E    D0      099A  2790 150$:    MOVL    (SP)+,R0                    ; Get buffer address
            08    13      099D  2791          BEQL    155$                        ; Br if end of list
 00000000'GF     16      099F  2792          JSB     G^EXE$DEANONPAGED           ; Deallocate the block
            F3    11      09A5  2793          BRB     150$                        ; Try for more
                        09A7  2794 155$:    POPQ    R3                          ; Restore R3, R4
                        09AA  2795
          1D04    30      09AA  2796          BSBW    ADD_MULTI                   ; Compile a new multicast address list
        33 50     E8      09AD  2797          BLBS    R0,T90$                     ; Br if all okay
                        09B0  2798
                        09B0  2799          ASSUME  NMA$C_STATE_ON EQ 0
                        09B0  2800          ASSUME  NMA$C_STATE_OFF EQ 1
                        09B0  2801
  04 00DA C5    E8      09B0  2802          BLBS    UCB$B_XQ_PRM(R5),160$       ; Br if NOT a PROMISCUOUS user
       0214 C4  D4      09B5  2803          CLRL    CDB_L_PRMUSER(R4)           ; Else, clear the PROMISCUOUS user addr
                        09B9  2804 160$:    CLRBIT  #UCB$V_XQ_INITED,-          ; Indicate unit is not initialized
                        09B9  2805                  UCB$W_DEVSTS(R5)            ;       ...
  20 A6    57    C0      09BE  2806          ADDL    R7,JIB$L_BYTCNT(R6)         ; Restore quota
  24 A6    57    C0      09C2  2807          ADDL    R7,JIB$L_BYTLM(R6)          ; ..and byte limit
51    0B0F  8F    3C      09C6  2808 170$:    MOVZWL  #NMA$C_PCLI_MCA,R1          ; Indicate bad multicast address
         FF62   31      09CB  2809          BRW     120$                        ; Return in error
                        09CE  2810 ;
                        09CE  2811 ; Change mode request - might have to reset multicast address list
                        09CE  2812 ;
                        09CE  2813 180$:    ASSUME  UCB$V_XQ_INITED  EQ  0
     03 68 A5   E8      09CE  2814          BLBS    UCB$W_DEVSTS(R5),185$       ; Br if unit inited
         FE07   31      09D2  2815          BRW     10$                         ; Else, success
                        09D5  2816
                        09D5  2817 185$:    SETIPL  UCB$B_FIPL(R5)              ; Sync access to UCB and CDB
          1CD5    30    09D9  2818          BSBW    ADD_MULTI                   ; Compile global multicast address list
        E7 50     E9      09DC  2819          BLBC    R0,T70$                     ; Br if error
  21 A3    06    90      09DF  2820          MOVB    #XQ_FC_V_CHMODE,IRP$B_XQ_FUNC(R3) ; Set function request
         02DD   31      09E3  2821 190$:    BRW     QUEPKT                      ; Queue packet to driver
```

```
                        09E6  2823 ;+
                        09E6  2824 ; Take quota subroutine
                        09E6  2825 ;
                        09E6  2826 ;       Calculate buffer quota and check against user's quota
                        09E6  2827 ;
                        09E6  2828 ; Inputs:       R7 = Scratch
                        09E6  2829 ;               R6 = Scratch
                        09E6  2830 ;               R5 = UCB address
                        09E6  2831 ;               R4 = PCB address
                        09E6  2832 ;
                        09E6  2833 ;
                        09E6  2834 ; Outputs:      R7 = Quota taken
                        09E6  2835 ;               R6 = JIB address
                        09E6  2836 ;               R5 = UCB address
                        09E6  2837 ;               R4 = PCB address
                        09E6  2838 ;               R0 = Status
                        09E6  2839 ;
                        09E6  2840 ;       R1,R2 are destroyed.
                        09E6  2841 ;
                        09E6  2842 ; Implicit outputs:
                        09E6  2843 ;
                        09E6  2844 ;       BR to ABORTIO if quota is exceeded.
                        09E6  2845 ;
                        09E6  2846 ;-
                        09E6  2847 TAKE_QUOTA:
          50    D4      09E6  2848         CLRL    R0                          ; Assume failure
     00D6 C5    B6      09E8  2849         INCW    UCB$W_XQ_BSZ(R5)            ; Round buffer size to even value
00D6 C5    01    AA      09EC  2850         BICW    #1,UCB$W_XQ_BSZ(R5)         ;       ...
51   00D5 C5    9A      09F1  2851         MOVZBL  UCB$B_XQ_BFN(R5),R1          ; Get number of receive buffers
  52   42 A5    3C      09F6  2852         MOVZWL  UCB$W_DEVBUFSIZ(R5),R2       ; Get buffer size
     51   52    C4      09FA  2853         MULL    R2,R1                       ; Get needed quota
     57   51    3C      09FD  2854         MOVZWL  R1,R7                       ; Copy quota
     57   51    D1      0A00  2855         CMPL    R1,R7                       ; Overflow?
          30    12      0A03  2856         BNEQ    110$                        ; Br if yes - error
 7E    24 A4    D0      0A05  2857         MOVL    PCB$L_STS(R4),-(SP)         ; Save current PCB status
                0A09  2858         SETBIT  #PCB$V_SSRWAIT,-            ; Do not go into a resource wait
                0A09  2859                 PCB$L_STS(R4)               ;   just to check the quota
          53    DD      0A0E  2860         PUSHL   R3                          ; Save R3
00000000'GF     16      0A10  2861         JSB     G^EXE$BUFQUOPRC             ; Check quota
          53 8ED0       0A16  2862         POPL    R3                          ; Restore R3
  24 A4    8E    D0      0A19  2863         MOVL    (SP)+,PCB$L_STS(R4)         ; Restore previous PCB status
       02   50    E8      0A1D  2864         BLBS    R0,50$                      ; Br if success
          10    11      0A20  2865         BRB     80$                         ; Else, return error
                0A22  2866
56    0080 C4    D0      0A22  2867 50$:    MOVL    PCB$L_JIB(R4),R6            ; Get JIB address
  20 A6    57    C2      0A27  2868         SUBL    R7,JIB$L_BYTCNT(R6)         ; Adjust quota
  24 A6    57    C2      0A2B  2869         SUBL    R7,JIB$L_BYTLM(R6)          ; ..and byte limit
     50    01    9A      0A2F  2870         MOVZBL  #1,R0                       ; Indicate success
          51    D4      0A32  2871 80$:    CLRL    R1                          ; No error return
          05    0A34  2872 90$:    RSB
                0A35  2873
     50    14    9A      0A35  2874 110$:   MOVZBL  #SS$_BADPARAM,R0            ; Setup error code
51   0451 8F    3C      0A38  2875         MOVZWL  #NMA$C_PCLI_BFN,R1          ; Assume BAD BFN
          F5    11      0A3D  2876         BRB     90$                         ; Return error
                0A3F  2877
                0A3F  2878 ;+
                0A3F  2879 ; CHECK_QUOTA - check SHARED unit's quota
```

```
                        0A3F  2880 ;
                        0A3F  2881 ; Inputs:       R9 = Scratch
                        0A3F  2882 ;               R7 = Original UCB address
                        0A3F  2883 ;               R5 = UCB address
                        0A3F  2884 ;               R4 = PCB address
                        0A3F  2885 ;
                        0A3F  2886 ;
                        0A3F  2887 ; Outputs:      R9 = Quota taken
                        0A3F  2888 ;               R7 = Original UCB address
                        0A3F  2889 ;               R5 = UCB address
                        0A3F  2890 ;               R4 = PCB address
                        0A3F  2891 ;               R0 = Status
                        0A3F  2892 ;
                        0A3F  2893 ;       R1,R2 are destroyed.
                        0A3F  2894 ;
                        0A3F  2895 ;-
                        0A3F  2896 CHECK_QUOTA:
                        0A3F  2897          PUSHQ   R6                          ; Save R6, R7
              55  DD    0A42  2898          PUSHL   R5                          ; Save UCB address
        55    57  D0    0A44  2899          MOVL    R7,R5                       ; Copy original UCB address
              9D  10    0A47  2900          BSBB    TAKE_QUOTA                  ; Charge quota to user
              55  8ED0  0A49  2901          POPL    R5                          ; Restore UCB address
          16  50  E9    0A4C  2902          BLBC    R0,90$                      ; Br if error
        59    57  D0    0A4F  2903          MOVL    R7,R9                       ; Copy quota taken
51      018B  C5  3C    0A52  2904          MOVZWL  UCB$W_XQ_TOTQUO(R5),R1      ; Get total quota
        51    59  C0    0A57  2905          ADDL    R9,R1                       ; Compute new total
        50    51  3C    0A5A  2906          MOVZWL  R1,R0                       ; Copy quota
        50    51  D1    0A5D  2907          CMPL    R1,R0                       ; Overflow?
              07  12    0A60  2908          BNEQ    110$                        ; Br if yes, error
        50    01  9A    0A62  2909          MOVZBL  #1,R0                       ; Else, return success
                        0A65  2910 90$:     POPQ    R6                          ; Retore R6, R7
              05  0A68  2911          RSB                                 ; Return to caller
                        0A69  2912
20 A6   59    C0  0A69  2913 110$:  ADDL    R9,JIB$L_BYTCNT(R6)         ; Restore quota
24 A6   59    C0  0A6D  2914        ADDL    R9,JIB$L_BYTLM(R6)          ; ..and byte limit
        50    1C  3C    0A71  2915        MOVZWL  #SS$_EXQUOTA,R0             ; Return bad quota
              51  D4    0A74  2916        CLRL    R1                          ; No parameter return
              ED  11    0A76  2917        BRB     90$                         ; Exit with error
                        0A78  2918
```

```
                        0A78  2920          .SBTTL  SHR_UCB - CREATE SHARED UCB
                        0A78  2921 ;++
                        0A78  2922 ; SHR_UCB - CREATE SHARED UCB
                        0A78  2923 ;
                        0A78  2924 ; Functional description:
                        0A78  2925 ;
                        0A78  2926 ; This subroutine creates a shared UCB if this is the first SHARED user of the
                        0A78  2927 ; particular protocol type.  Else, the already created SHARED UCB is found and
                        0A78  2928 ; a shared data structure is added to the list of shared users of that protocol
                        0A78  2929 ; type.
                        0A78  2930 ;
                        0A78  2931 ; Inputs:
                        0A78  2932 ;
                        0A78  2933 ;       R3 = IRP address
                        0A78  2934 ;       R4 = PCB address
                        0A78  2935 ;       R5 = UCB address
                        0A78  2936 ;       R6 = CCB address
                        0A78  2937 ;       R7 = Function code
                        0A78  2938 ;       AP = Address of first function-dependent QIO parameter
                        0A78  2939 ;
                        0A78  2940 ;       IPL = FIPL
                        0A78  2941 ;
                        0A78  2942 ; Outputs:
                        0A78  2943 ;
                        0A78  2944 ;       R0 = Status return for request
                        0A78  2945 ;       R1 = Bad parameter code (if bad parameter error code)
                        0A78  2946 ;       R2,R7 are destroyed.
                        0A78  2947 ;       All other registers are preserved.
                        0A78  2948 ;
                        0A78  2949 ;       IPL = ASTDEL
                        0A78  2950 ;
                        0A78  2951 ;--
                        0A78  2952
                        0A78  2953 SHR_UCB::                            ; Setup shared protocol UCB
     0300 8F    BB      0A78  2954          PUSHR   #^M<R8,R9>          ; Save registers
 00000000'GF    16      0A7C  2955          JSB     G^SCH$IOLOCKW       ; Lock I/O data base for write access
     57    55    D0      0A82  2956          MOVL    R5,R7               ; Save UCB address
50   21C4 8F    3C      0A85  2957          MOVZWL  #SS$_DUPUNIT,R0     ; Assume 2 channels assigned to UCB
  5C A5    01    B1      0A8A  2958          CMPW    #1,UCB$W_REFC(R5)   ; Is there only one reference to UCB?
           75    12      0A8E  2959          BNEQ    23$                 ; Br if not, error (more than 1 channel)
58   00D4 C5    9A      0A90  2960          MOVZBL  UCB$B_XQ_ACC(R5),R8 ; Save access mode
           03    E0      0A95  2961          BBS     #UCB$V_XQ_SHARE,-   ; Br if we are already the SHARED UCB
     2C 68 A5          0A97  2962                   UCB$W_DEVSTS(R5),10$ ;
           54    DD      0A9A  2963          PUSHL   R4                  ; Save PCB address
  54   24 A5    D0      0A9C  2964          MOVL    UCB$L_CRB(R5),R4    ; Get CRB address
  54   10 A4    D0      0AA0  2965          MOVL    CRB$L_AUXSTRUC(R4),R4 ; Get CDB address
51   00CA C5    B0      0AA4  2966          MOVW    UCB$W_XQ_PROTYP(R5),R1 ; Get protocol type
         221A    30      0AA9  2967          BSBW    MATCH_PROTYP        ; Try to match protocol type
           54 8ED0       0AAC  2968          POPL    R4                  ; Restore PCB address
       14  50    E8      0AAF  2969          BLBS    R0,10$              ; Br if found
     55    57    D0      0AB2  2970          MOVL    R7,R5               ; Else, get back old UCB address
       2C A5    D4      0AB5  2971          CLRL    UCB$L_PID(R5)       ; Make this UCB shareable
  50   1C A5    D0      0AB8  2972          MOVL    UCB$L_ORB(R5),R0    ; Get the ORB address
           60    D4      0ABC  2973          CLRL    ORB$L_OWNER(R0)     ;  clear owner UIC as well
     00C4 C5    D4      0ABE  2974          CLRL    UCB$L_XQ_DEFUSR(R5) ; Clear default user
           08    A8      0AC2  2975          BISW    #UCB$M_XQ_SHARE,-   ; Indicate that UCB is in SHARED mode
       68 A5          0AC4  2976                   UCB$W_DEVSTS(R5)    ;
```

```
                        0AC6  2977 ;
                        0AC6  2978 ; Allocate a share data structure and link it in
                        0AC6  2979 ;
      58   02   91      0AC6  2980 10$:     CMPB    #NMA$C_ACC_LIM,R8       ; Is the new user a limited user?
           0B   13      0AC9  2981          BEQL    20$                     ; Br if yes - okay
50    0840 8F   3C      0ACB  2982          MOVZWL  #SS$_DEVALLOC,R0        ; Assume protocol already allocated
      00C4 C5   D5      0AD0  2983          TSTL    UCB$L_XQ_DEFUSR(R5)     ; Is there already a default user?
           2F   12      0AD4  2984          BNEQ    23$                     ; Br if YES - error
             00C8   30  0AD6  2985 20$:     BSBW    CHECK_PARAM             ; Check out all parameters
           2B 50   E9  0AD9  2986          BLBC    R0,25$                  ; Br on error
             FF60  30  0ADC  2987          BSBW    CHECK_QUOTA             ; Check our quota
           25 50   E9  0ADF  2988          BLBC    R0,25$                  ; Br if error
      51   2A   9A      0AE2  2989          MOVZBL  #SHR_C_LENGTH,R1        ; Get size of structure to allocate
           53   DD      0AE5  2990          PUSHL   R3                      ; Save IRP address
 00000000'GF   16      0AE7  2991          JSB     G^EXE$ALLOCBUF          ; Allocate buffer, reset IPL to ASTDEL
           53 8ED0  0AED  2992          POPL    R3                      ; Restore IRP address
         1A 50   E8  0AF0  2993          BLBS    R0,30$                  ; Br if success
50    0124 8F   3C      0AF3  2994          MOVZWL  #SS$_INSFMEM,R0         ; Else, return error reason
51    0080 C4   D0      0AF8  2995          MOVL    PCB$L_JIB(R4),R1        ; Get JIB address
   20 A1   59   C0      0AFD  2996          ADDL    R9,JIB$L_BYTCNT(R1)     ; Restore quota
   24 A1   59   C0      0B01  2997          ADDL    R9,JIB$L_BYTLM(R1)      ; ..and byte limit
           51   D4      0B05  2998 23$:     CLRL    R1                      ; No bad parameter code
      55   57   D0      0B07  2999 25$:     MOVL    R7,R5                   ; Get back the OLD UCB address
             0085   31  0B0A  3000          BRW     80$                     ; Exit with error
                        0B0D  3001 ;
                        0B0D  3002 ; Initialize shared (SHR) data structure
                        0B0D  3003 ;
                        0B0D  3004 30$:     ASSUME  SHR_L_QFL EQ 0
                        0B0D  3005          ASSUME  SHR_L_QBL EQ SHR_L_QFL+4
           82   7C      0B0D  3006          CLRQ    (R2)+                   ; Zero LINK pointers
                        0B0F  3007          ASSUME  SHR_W_SIZE EQ SHR_L_QBL+4
      82   51   B0      0B0F  3008          MOVW    R1,(R2)+                ; Save size
                        0B12  3009          ASSUME  SHR_B_TYPE EQ SHR_W_SIZE+2
           52   D6      0B12  3010          INCL    R2                      ; Filled by EXE$ALLOCBUF routine
                        0B14  3011          ASSUME  SHR_B_STS EQ SHR_B_TYPE+1
      82   01   90      0B14  3012          MOVB    #SHR_STS_M_INITED,(R2)+ ; Initialize SHR status
                        0B17  3013          ASSUME  SHR_L_PID EQ SHR_B_STS+1
   82   0C A3   D0      0B17  3014          MOVL    IRP$L_PID(R3),(R2)+     ; Save users PID and CHAN for
                        0B1B  3015          ASSUME  SHR_W_CHAN EQ SHR_L_PID+4
   82   28 A3   B0      0B1B  3016          MOVW    IRP$W_CHAN(R3),(R2)+    ;  for future lookups
                        0B1F  3017          ASSUME  SHR_G_DEST EQ SHR_W_CHAN+2
82    00CC C7   D0      0B1F  3018          MOVL    UCB$G_XQ_DES(R7),(R2)+  ; Save destination address
82    00D0 C7   B0      0B24  3019          MOVW    UCB$G_XQ_DES+4(R7),(R2)+ ;
                        0B29  3020          ASSUME  SHR_Q_QUEUES EQ SHR_G_DEST+6
      51   02   9A      0B29  3021          MOVZBL  #SHR_C_QUEUES,R1        ; Get number of queues in structure
      82   62   DE      0B2C  3022 40$:     MOVAL   (R2),(R2)+              ; Set forward link pointer
   82   FC   A2   D0    0B2F  3023          MOVL    -4(R2),(R2)+            ; Set backward link pointer
        F6 51   F5      0B33  3024          SOBGTR  R1,40$                  ; Loop if more listheads
                        0B36  3025          ASSUME  SHR_W_QUOTA EQ SHR_Q_QUEUES+<8*SHR_C_QUEUES>
      82   59   B0      0B36  3026          MOVW    R9,(R2)+                ; Initialize quota
                        0B39  3027
                        0B39  3028          ASSUME  UCB$V_XQ_INITED EQ 0
   0C 68 A5   E9      0B39  3029          BLBC    UCB$W_DEVSTS(R5),50$    ; Br if UCB not initialized
00C8 C5   59   A0      0B3D  3030          ADDW    R9,UCB$W_XQ_QUOTA(R5)   ; Add to the current quota
018B C5   59   A0      0B42  3031          ADDW    R9,UCB$W_XQ_TOTQUO(R5)  ;  and the total quota
           0A   11      0B47  3032          BRB     55$                     ; Continue
00C8 C5   59   B0      0B49  3033 50$:     MOVW    R9,UCB$W_XQ_QUOTA(R5)   ; Set current quota
```

```
018B C5    59    B0   0B4E  3034                MOVW    R9,UCB$W_XQ_TOTQUO(R5)  ;  and total quota
       52    2A    C2   0B53  3035  55$:          SUBL    #SHR_C_LENGTH,R2        ; Backup to beginning of structure
       58    02    91   0B56  3036                CMPB    #NMA$C_ACC_LIM,R8       ; Is this for limited use?
             07    13   0B59  3037                BEQL    60$                     ; Br if YES
00C4 C5    52    D0   0B5B  3038                MOVL    R2,UCB$L_XQ_DEFUSR(R5)  ; Else, save default user address
             05    11   0B60  3039                BRB     65$                     ; Skip linking onto list
009C D5    62    0E   0B62  3040  60$:          INSQUE  (R2),@UCB$Q_XQ_SHARE+4(R5) ; Link user into shared user list
         5C A5    B6   0B67  3041  65$:          INCW    UCB$W_REFC(R5)          ; Increment the ref count on the
                       0B6A  3042                                               ;  UCB to be used
       57    55    D1   0B6A  3043                CMPL    R5,R7                   ; Was this the original UCB?
             20    13   0B6D  3044                BEQL    70$                     ; Br if YES - no more work to do
         5C A5    B6   0B6F  3045                INCW    UCB$W_REFC(R5)          ; Else, increment REFC (for $DASSGN)
    1C A3    55    D0   0B72  3046                MOVL    R5,IRP$L_UCB(R3)        ; Return new UCB address
       66    55    D0   0B76  3047                MOVL    R5,CCB$L_UCB(R6)        ;  in CCB also.
             28    BB   0B79  3048                PUSHR   #^M<R3,R5>              ; Save IRP, real UCB address
       55    57    D0   0B7B  3049                MOVL    R7,R5                   ; Copy old UCB address
         5C A5    B7   0B7E  3050                DECW    UCB$W_REFC(R5)          ; Decrement the reference count
00000000'GF    16   0B81  3051                JSB     G^IOC$CREDIT_UCB        ; Restore UCB quota to JIB
00000000'GF    16   0B87  3052                JSB     G^IOC$DELETE_UCB        ; Delete the old UCB
             28    BA   0B8D  3053                POPR    #^M<R3,R5>              ; Restore IRP, UCB address
       50    01    9A   0B8F  3054  70$:          MOVZBL  S^#SS$_NORMAL,R0        ; Return success
       0300  8F    BA   0B92  3055  80$:          POPR    #^M<R8,R9>              ; Restore registers, R4 is PCB address
             0B    BB   0B96  3056                PUSHR   #^M<R0,R1,R3>           ; Save IRP address, status return
00000000'GF    16   0B98  3057                JSB     G^SCH$IOUNLOCK          ; Unlock I/O data base
             0B    BA   0B9E  3058                POPR    #^M<R0,R1,R3>           ; Restore IRP address, status return
             05    0BA0  3059                RSB                             ; Return to caller
```

```
                        0BA1  3061          .SBTTL  CHECK_PARAM - CHECK SHARED USERS PARAMETERS
                        0BA1  3062 ;++
                        0BA1  3063 ; CHECK_PARAM - CHECK SHARED USERS PARAMETERS
                        0BA1  3064 ;
                        0BA1  3065 ; Functional description:
                        0BA1  3066 ;
                        0BA1  3067 ; Validate all parameters between the requesting SHARED user and the old
                        0BA1  3068 ; existing SHARED user to make sure that are the same.
                        0BA1  3069 ;
                        0BA1  3070 ; Inputs:
                        0BA1  3071 ;       R5 = UCB address of existing shared user
                        0BA1  3072 ;       R7 = UCB address of new shared user
                        0BA1  3073 ;       R8 = Protocol access mode
                        0BA1  3074 ;
                        0BA1  3075 ; Outputs:
                        0BA1  3076 ;       R0 = Status of request
                        0BA1  3077 ;       R1 = Bad parameter code if validation failed
                        0BA1  3078 ;
                        0BA1  3079 ;--
                        0BA1  3080
                        0BA1  3081 CHECK_PARAM:                                 ; Check user parameters
          56    DD      0BA1  3082          PUSHL   R6                          ; Save registers
    50    00D6  C5  9E  0BA3  3083          MOVAB   UCB$B_XQ_SHRPRM(R5),R0      ; Get address of current parameters
    56    00D6  C7  9E  0BA8  3084          MOVAB   UCB$B_XQ_SHRPRM(R7),R6      ; Get address of new parameters
          52    08  9A  0BAD  3085          MOVZBL  #UCB$C_XQ_SHRPRM,R2         ; Set size of parameter list
                        0BB0  3086 ;
                        0BB0  3087 ; Validate all user settable parameters except for Physical address
                        0BB0  3088 ;
          66    60  91  0BB0  3089 10$:     CMPB    (R0),(R6)                   ; Match?
                2B  12  0BB3  3090          BNEQ    70$                         ; Br if no
          86    80  90  0BB5  3091          MOVB    (R0)+,(R6)+                 ; Store current value in UCB
            F5  52  F5  0BB8  3092          SOBGTR  R2,10$                      ; Loop if more to check
                        0BBB  3093 ;
                        0BBB  3094 ; NOW, check if user has given a hardware physical address.
                        0BBB  3095 ;
                        0BBB  3096 ;       R0 = Address of parameters in UCB
                        0BBB  3097 ;       R6 = Address of parameters in CDB
                        0BBB  3098 ;
                        0BBB  3099          ASSUME  CDB_G_PHA EQ CDB_B_CON+1
                        0BBB  3100          ASSUME  UCB$G_XQ_PHA EQ UCB$B_XQ_CON+1
    66    FFFFFFFF  8F  D1  0BBB  3101          CMPL    #-1,(R6)                ; Is user physical address defined?
                08  12  0BC2  3102          BNEQ    20$                         ; Br if yes
     04 A6    FFFF  8F  B1  0BC4  3103          CMPW    #-1,4(R6)               ; Is user physical address defined?
                0F  13  0BCA  3104          BEQL    30$                         ; Br if not
        51    0B04  8F  3C  0BCC  3105 20$:     MOVZWL  #NMA$C_PCLI_PHA,R1      ; Assume bad physical address
              86    80  D1  0BD1  3106          CMPL    (R0)+,(R6)+             ; Physical address match??
                10  12  0BD4  3107          BNEQ    80$                         ; Br if no
              86    80  B1  0BD6  3108          CMPW    (R0)+,(R6)+             ; Still match??
                0B  12  0BD9  3109          BNEQ    80$                         ; Br if no
                        0BDB  3110 ;
                        0BDB  3111 ; If this is the shared default user, then set the multicast address list.
                        0BDB  3112 ;
                  000F  30  0BDB  3113 30$:     BSBW    SET_MULTIN              ; Set new multicast address list
                    09  11  0BDE  3114          BRB     100$                    ; Exit
                        0BE0  3115 ;
                        0BE0  3116 ; Error on parameter validation
                        0BE0  3117 ;
```

```
                51   F5BF CF42    3C  0BE0  3118 70$:    MOVZWL  BAD_PARAM_TBL-2[R2],R1  ; Return parameter code
                     50      14    9A  0BE6  3119 80$:    MOVZBL  S^#SS$_BADPARAM,R0     ; Return bad parameter error
                             56 8ED0  0BE9  3120 100$:   POPL    R6                      ; Restore registers
                                05  0BEC  3121            RSB
```

```
                     0BED  3123          .SBTTL  SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
                     0BED  3124 ;++
                     0BED  3125 ; SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
                     0BED  3126 ;
                     0BED  3127 ; Functional description:
                     0BED  3128 ;
                     0BED  3129 ; Copy the multicast address list from the old UCB to the new UCB.
                     0BED  3130 ; This operation is only done for the SHARED DEFAULT user of the PROTOCOL.
                     0BED  3131 ;
                     0BED  3132 ; Inputs:
                     0BED  3133 ;       R5 = UCB address of existing shared user
                     0BED  3134 ;       R7 = UCB address of new shared user
                     0BED  3135 ;       R8 = Protocol access mode
                     0BED  3136 ;
                     0BED  3137 ; Outputs:
                     0BED  3138 ;       R0 = Always success
                     0BED  3139 ;       All other are preserved.
                     0BED  3140 ;
                     0BED  3141 ;--
                     0BED  3142
                     0BED  3143 SET_MULTIN:                                  ; Set new multicast list
           58  01  91  0BED  3144          CMPB    #NMA$C_ACC_SHR,R8         ; Is this the shared default user?
               15  12  0BF0  3145          BNEQ    50$                       ; Br if not
               3E  BB  0BF2  3146          PUSHR   #^M<R1,R2,R3,R4,R5>       ; Save registers
00E7 C5  00E7 C7  0048 8F  28  0BF4  3147          MOVC3   #6*MAX_C_MLT,UCB$G_XQ_MULTI(R7), - ; Copy multicast list
                           0BFE  3148                  UCB$G_XQ_MULTI(R5)    ;
               3E  BA  0BFE  3149          POPR    #^M<R1,R2,R3,R4,R5>       ; Restore registers
           00E5 C7  90  0C00  3150          MOVB    UCB$B_XQ_MULTI(R7),-      ; Copy the number of valid addresses
           00E5 C5      0C04  3151                  UCB$B_XQ_MULTI(R5)        ;
           50  01  9A  0C07  3152 50$:      MOVZBL  #SS$_NORMAL,R0            ; Return success
               05  0C0A  3153          RSB                                   ; Return to caller
```

```
                        0C0B  3155          .SBTTL  SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
                        0C0B  3156 ;++
                        0C0B  3157 ; SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
                        0C0B  3158 ;
                        0C0B  3159 ; Functional description:
                        0C0B  3160 ;
                        0C0B  3161 ; Process read status and read counters requests.
                        0C0B  3162 ;
                        0C0B  3163 ; The QIO parameters for SENSEMODE are:
                        0C0B  3164 ;
                        0C0B  3165 ;       P1 = Optional address of quadword buffer
                        0C0B  3166 ;       P2 = Optional address of buffer descriptor for extended characteristics
                        0C0B  3167 ;
                        0C0B  3168 ; The SUBFUNCTION modifiers are as follows:
                        0C0B  3169 ;
                        0C0B  3170 ; 1)    READ PARAMETERS -- NO MODIFIER.
                        0C0B  3171 ;       This function reads the QNA parameters and returns them to the user.
                        0C0B  3172 ;
                        0C0B  3173 ; 2)    READ COUNTERS -- IO$M_RD_COUNT SET.
                        0C0B  3174 ;       This function reads the QNA counters and returns them to the user.
                        0C0B  3175 ;
                        0C0B  3176 ;       CLEAR COUNTERS -- IO$M_CLR_COUNT SET.
                        0C0B  3177 ;       This modifier must be used with the read counters modifier to clear
                        0C0B  3178 ;       the counters as they are read.
                        0C0B  3179 ;
                        0C0B  3180 ;
                        0C0B  3181 ; Inputs:
                        0C0B  3182 ;
                        0C0B  3183 ;       R3 = IRP address
                        0C0B  3184 ;       R4 = PCB address
                        0C0B  3185 ;       R5 = UCB address
                        0C0B  3186 ;       R6 = CCB address
                        0C0B  3187 ;       R7 = Function code
                        0C0B  3188 ;       AP = Address of first function-dependent QIO parameter
                        0C0B  3189 ;
                        0C0B  3190 ;       IPL = ASTDEL
                        0C0B  3191 ;
                        0C0B  3192 ; Outputs:
                        0C0B  3193 ;
                        0C0B  3194 ;       R0 = Status return of SENSEMODE request
                        0C0B  3195 ;
                        0C0B  3196 ;       R1,R2,R6,R7 are destroyed.
                        0C0B  3197 ;
                        0C0B  3198 ;--
                        0C0B  3199
                        0C0B  3200 SENSEMODE_FDT::                              ; SENSE MODE I/O FDT processing
50   0084 8F    3C      0C0B  3201          MOVZWL  #SS$_DEVOFFLINE,R0          ; Assume unit if offline
           04    E1      0C10  3202          BBC     #UCB$V_ONLINE,-             ; Br if unit not online
     23 64 A5            0C12  3203                  UCB$W_STS(R5),15$           ;
57    20 A3    B0      0C15  3204          MOVW    IRP$W_FUNC(R3),R7           ; Get entire function code
14 57    09    E0      0C19  3205          BBS     #IO$V_CTRL,R7,10$           ; Br if line request
                        0C1D  3206 ;
                        0C1D  3207 ; Check if read circuit counters
                        0C1D  3208 ;
03 57    08    E1      0C1D  3209          BBC     #IO$V_RD_COUNT,R7,5$        ; Br if not read circuit counters
      00D8    31      0C21  3210          BRW     READ_CIRC_CTR               ; Else, get the circuit counters
   50    01    9A      0C24  3211 5$:      MOVZBL  S^#SS$_NORMAL,R0            ; Return success
```

```
                 51    44 A5    D0   0C27  3212          MOVL    UCB$L_DEVDEPEND(R5),R1  ; Get device dependent information
           00000000'GF    17   0C2B  3213          JMP     G^EXE$FINISHIO          ; Complete the I/O request
                                0C31  3214
                 06 57    08   E1   0C31  3215  10$:    BBC     #IO$V_RD_COUNT,R7,20$   ; Br if not read counters
                                0C35  3216  ;
                                0C35  3217  ; Read counters - modifier RD_COUNT
                                0C35  3218  ;
                       00A2     31   0C35  3219          BRW     READ_LINE_CTR           ; Get the line counters
                                0C38  3220
                       F70B     31   0C38  3221  15$:    BRW     ABORTIO                 ; Abort the I/O request
                                0C3B  3222  ;
                                0C3B  3223  ; Read parameters - no modifier
                                0C3B  3224  ;
                    44 A3       B4   0C3B  3225  20$:    CLRW    IRP$W_XQ_P2SIZ(R3)      ; No return data
                 54    24 A5    D0   0C3E  3226          MOVL    UCB$L_CRB(R5),R4        ; Get CRB address
                 54    10 A4    D0   0C42  3227          MOVL    CRB$L_AUXSTRUC(R4),R4   ; Get CDB address
                       51 08    3C   0C46  3228          MOVZWL  S^#8,R1                 ; Size of P1 buffer if present
                          0181  30   0C49  3229          BSBW    CHECK_BUFS              ; Check P1 and P2 buffers
                       50 01    9A   0C4C  3230          MOVZBL  S^#SS$_NORMAL,R0        ; Assume success
                 3A A3    50    B0   0C4F  3231          MOVW    R0,IRP$W_XQ_STATUS(R3)  ;      ...
                 38 A3    51    B0   0C53  3232          MOVW    R1,IRP$W_XQ_USERSIZ(R3) ; Save user P2 buffer length
                          47    13   0C57  3233          BEQL    40$                     ; Br if no P2 buffer present
                                0C59  3234
                 40 A3    52    D0   0C59  3235          MOVL    R2,IRP$L_XQ_P2BUF(R3)   ; Save user P2 buffer address
                       1D38     30   0C5D  3236          BSBW    RETURN_P2               ; Return the P2 parameters
                 44 A3    50    B0   0C60  3237          MOVW    R0,IRP$W_XQ_P2SIZ(R3)   ; Set size of return data
                          54    D5   0C64  3238          TSTL    R4                      ; Is CDB present?
                          2C    13   0C66  3239          BEQL    30$                     ; Br if no - okay to return now
              51    50    0A    C1   0C68  3240          ADDL3   S^#10,R0,R1             ; Check if default physical
                                0C6C  3241                                            ;  address can fit
                    38 A3    51    B1   0C6C  3242          CMPW    R1,IRP$W_XQ_USERSIZ(R3)       ; Is buffer big enough for
                                0C70  3243                                            ; default physical address?
                          1C    1A   0C70  3244          BGTRU   25$                     ; Br if no
                    44 A3    0A    A0   0C72  3245          ADDW    #10,IRP$W_XQ_P2SIZ(R3)  ; Add DPA to return buffer size
                 52 40 A3    50    C1   0C76  3246          ADDL3   R0,IRP$L_XQ_P2BUF(R3),R2      ; Get buffer address
                                0C7B  3247                                            ; past end of return data
              82 00061488 8F    D0   0C7B  3248          MOVL    #<6a16>+NMA$C_PCLI_HWA!-; Store parameter code + size
                                0C82  3249                  PRM_TYP_M_STRING,(R2)+  ;  and indicate this is a string
                 82    0254 C4    D0   0C82  3250          MOVL    CDB_G_HWA(R4),(R2)+     ; Store Default Physical Address
                 82    0258 C4    B0   0C87  3251          MOVW    CDB_G_HWA+4(R4),(R2)+   ;
                          06    11   0C8C  3252          BRB     30$                     ; All is okay
              3A A3    0601 8F    B0   0C8E  3253  25$:    MOVW    #SS$_BUFFEROVF,IRP$W_XQ_STATUS(R3) ; Return partial success
                    50    44 A3    3C   0C94  3254  30$:    MOVZWL  IRP$W_XQ_P2SIZ(R3),R0   ; Get size of user return data
                 50    50    10    78   0C98  3255          ASHL    #16,R0,R0               ; Shift size of buffer return
                    50    3A A3    B0   0C9C  3256          MOVW    IRP$W_XQ_STATUS(R3),R0  ; Get status
                                0CA0  3257
                          54    D5   0CA0  3258  40$:    TSTL    R4                      ; Is there a CDB?
                          15    13   0CA2  3259          BEQL    50$                     ; Br if no CDB yet!
                    52    3C A3    D0   0CA4  3260          MOVL    IRP$L_XQ_USERBUF(R3),R2 ; Retrieve P1 buffer address
                          0F    13   0CA8  3261          BEQL    50$                     ; Br if none
                    62    40 A5    7D   0CAA  3262          MOVQ    UCB$B_DEVCLASS(R5),(R2) ; Else, return characteristics
              04 A2    0114 C4    C8   0CAE  3263          BISL    CDB_L_DEVDEPEND(R4),4(R2) ;      ...
                 51    0114 C4    D0   0CB4  3264          MOVL    CDB_L_DEVDEPEND(R4),R1  ; Get device dependent info
                    51    44 A5    C8   0CB9  3265  50$:    BISL    UCB$L_DEVDEPEND(R5),R1  ;  ..from UCB also
                     00000000'GF    17   0CBD  3266          JMP     G^EXE$FINISHIO          ; Complete the I/O request
                                0CC3  3267
                                0CC3  3268  ;
```

```
                     0CC3  3269 ; Queue I/O request to driver
                     0CC3  3270 ;
                     0CC3  3271 QUEPKT:                                      ; Queue packet to driver
                     0CC3  3272          SETIPL  UCB$B_FIPL(R5)              ; Raise IPL to fork IPL
00000000'GF    16    0CC7  3273          JSB     G^IOC$INITIATE              ; Intiate the I/O request
00000000'GF    17    0CCD  3274          JMP     G^EXE$QIORETURN             ; Lower IPL, and RET
```

```
                     OCD3  3276          .SBTTL  READ_LINE_CTR - READ THE LINE COUNTERS
                     OCD3  3277          .SBTTL  READ_CIRC_CTR - READ THE CIRCUIT COUNTERS
                     OCD3  3278 ;++
                     OCD3  3279 ; READ_LINE_CTR - READ THE LINE COUNTERS
                     OCD3  3280 ; READ_CIRC_CTR - READ THE CIRCUIT COUNTERS
                     OCD3  3281 ;
                     OCD3  3282 ; Functional description:
                     OCD3  3283 ;
                     OCD3  3284 ; Process read circuit counters request.
                     OCD3  3285 ;
                     OCD3  3286 ; The QIO parameters for SENSEMODE are:
                     OCD3  3287 ;
                     OCD3  3288 ;       P2 = Address of buffer descriptor for counters
                     OCD3  3289 ;
                     OCD3  3290 ;
                     OCD3  3291 ; Inputs:
                     OCD3  3292 ;
                     OCD3  3293 ;       R3 = IRP address
                     OCD3  3294 ;       R4 = PCB address
                     OCD3  3295 ;       R5 = UCB address
                     OCD3  3296 ;       R6 = CCB address
                     OCD3  3297 ;       R7 = Function code and modifier bits
                     OCD3  3298 ;       AP = Address of first function-dependent QIO parameter
                     OCD3  3299 ;
                     OCD3  3300 ; Outputs:
                     OCD3  3301 ;
                     OCD3  3302 ;       R0 = Status return of SENSEMODE request
                     OCD3  3303 ;
                     OCD3  3304 ;       R1,R2,R6,R7 are destroyed.
                     OCD3  3305 ;
                     OCD3  3306 ;--
                     OCD3  3307          .ENABL  LSB
                     OCD3  3308 ABORT_IRP:
     0700 8F    BA   OCD3  3309          POPR    #^M<R8,R9,R10>          ; Restore  registers
        F66C    31   OCD7  3310          BRW     ABORTIO                 ; Abort the I/O request
                     OCDA  3311
                     OCDA  3312 READ_LINE_CTR:                           ; Read the line counters
     0700 8F    BB   OCDA  3313          PUSHR   #^M<R8,R9,R10>          ; Save registers
  58 F427 CF    9E   OCDE  3314          MOVAB   LINE_CTR,R8             ; Get address of counter format table
     59   14    3C   OCE3  3315          MOVZWL  #LINE_CTR_SIZE,R9       ; Get number of entries in table
  56 006A 8F    3C   OCE6  3316          MOVZWL  #LINE_CTR_BUFSIZ,R6     ; Get size of system P2 buffer
   5A   24    A5   D0   OCEB  3317          MOVL    UCB$L_CRB(R5),R10       ; Get CRB address
  50 0124 8F    3C   OCEF  3318          MOVZWL  #SS$_INSFMEM,R0         ; Assume no CDB
   5A   10    AA   D0   OCF4  3319          MOVL    CRB$L_AUXSTRUC(R10),R10 ; Get CDB address
        D9    13   OCF8  3320          BEQL    ABORT_IRP               ; Br if none, abort the I/O
        12    11   OCFA  3321          BRB     10$                     ; Else, Continue in common code
                     OCFC  3322
                     OCFC  3323 READ_CIRC_CTR:                           ; Read the circuit counters
     0700 8F    BB   OCFC  3324          PUSHR   #^M<R8,R9,R10>          ; Save registers
  58 F455 CF    9E   OCD00 3325          MOVAB   CIRC_CTR,R8             ; Get address of counter format table
     59   06    3C   OCD05 3326          MOVZWL  #CIRC_CTR_SIZE,R9       ; Get number of entries in table
     56   20    3C   OCD08 3327          MOVZWL  #CIRC_CTR_BUFSIZ,R6     ; Get size of system P2 buffer
     5A   55    D0   OCD0B 3328          MOVL    R5,R10                  ; Use UCB for counters
                     OCD0E 3329
        00BE    30   OCD0E 3330 10$:     BSBW    CHECK_P2                ; Check the P2 buffer
     50   14    9A   OCD11 3331          MOVZBL  S^#SS$_BADPARAM,R0      ; Assume zero length buffer
  38 A3   51    B0   OCD14 3332          MOVW    R1,IRP$W_XQ_USERSIZ(R3) ; Save size of user P2 buffer
```

```
                B9   13   0D18  3333          BEQL    ABORT_IRP                       ; Br if no buffer
           51   56   D0   0D1A  3334          MOVL    R6,R1                           ; Get size of system P2 buffer
             00E9   30   0D1D  3335          BSBW    ALLOC_P2BUF                     ; Allocate the buffer
             B0 50   E9   0D20  3336          BLBC    R0,ABORT_IRP                    ; Br if error
      51     2C A3   D0   0D23  3337          MOVL    IRP$L_SVAPTE(R3),R1             ; Get system P2 buffer address
      04 A1     52   D0   0D27  3338          MOVL    R2,P2B_L_BUFFER(R1)             ; Save user P2 buffer address
           52   61   D0   0D2B  3339          MOVL    P2B_L_POINTER(R1),R2            ; Get address of data portion of buffer
                     0D2E  3340 ;
                     0D2E  3341 ; Get the counters kept by the driver
                     0D2E  3342 ;
           50   88   3C   0D2E  3343 20$:     MOVZWL  (R8)+,R0                        ; Get counter code
           82   50   B0   0D31  3344          MOVW    R0,(R2)+                        ; Return counter type code
           51   88   3C   0D34  3345          MOVZWL  (R8)+,R1                        ; Get offset word
           51   5A   C0   0D37  3346          ADDL    R10,R1                          ; Point to counter in UCB
50   50    03   0C   EF   0D3A  3347          EXTZV   #NMA$V_CNT_MAP,#3,R0,R0         ; Get width + bit map indicator
                     0D3F  3348          CASE    R0,TYPE=B,LIMIT=#2,<-           ; Dispatch on width and bit map
                     0D3F  3349                  30$,-                           ; 8 bit counter
                     0D3F  3350                  30$,-                           ; 8 bit counter + bit map
                     0D3F  3351                  40$,-                           ; 16 bit counter
                     0D3F  3352                  35$,-                           ; 16 bit counter + bit map
                     0D3F  3353                  35$>                            ; 32 bit counter
                     0D4D  3354
                     0D4D  3355 30$:     BUG_CHECK NOBUFPCKT,FATAL
                     0D51  3356
                     0D51  3357 ;
                     0D51  3358 ; 32 BIT counter/ 16 BIT counter + bitmap
                     0D51  3359 ;
           82   81   B0   0D51  3360 35$:     MOVW    (R1)+,(R2)+                     ; Store counter in buffer
      03 57     0A   E1   0D54  3361          BBC     #IO$V_CLR_COUNT,R7,40$          ; Br if not clear counter operation
             FE A1   B4   0D58  3362          CLRW    -2(R1)                          ; Else, clear the counter as well
                     0D5B  3363 ;
                     0D5B  3364 ; 16 BIT counter
                     0D5B  3365 ;
        82   61   B0   0D5B  3366 40$:     MOVW    (R1),(R2)+                      ; Store counter in buffer
   02 57     0A   E1   0D5E  3367          BBC     #IO$V_CLR_COUNT,R7,50$          ; Br if not clear counter operation
             61   B4   0D62  3368          CLRW    (R1)                            ; Else, clear the counter as well
         C7 59   F5   0D64  3369 50$:     SOBGTR  R9,20$                          ; Loop if more
      51     01   9A   0D67  3370          MOVZBL  S^#SS$_NORMAL,R1                ; Assume success
        38 A3   B1   0D6A  3371          CMPW    IRP$W_XQ_USERSIZ(R3),-          ; Is user's buffer big enough?
        32 A3        0D6D  3372                  IRP$W_BCNT(R3)                  ;
             0A   1E   0D6F  3373          BGEQU   60$                             ; Br if yes
        38 A3   B0   0D71  3374          MOVW    IRP$W_XQ_USERSIZ(R3),-          ; Else, set size to minimum
        32 A3        0D74  3375                  IRP$W_BCNT(R3)                  ; Of both
   51   0601 8F   B0   0D76  3376          MOVW    #SS$_BUFFEROVF,R1               ; Set partial success
     50   30 A3   D0   0D7B  3377 60$:     MOVL    IRP$W_BCNT-2(R3),R0             ; Get size of buffer returned in
                     0D7F  3378                                                  ; ..high word of R0
        50   51   B0   0D7F  3379          MOVW    R1,R0                           ; Get status return
      51   44 A5   D0   0D82  3380          MOVL    UCB$L_DEVDEPEND(R5),R1          ; Get device dependent info
           0700 8F   BA   0D86  3381          POPR    #^M<R8,R9,R10>                  ; Restore registers
 00000000'GF   17   0D8A  3382          JMP     G^EXE$FINISHIO                  ; Complete the I/O request
                     0D90  3383
                     0D90  3384          .DSABL  LSB
                     0D90  3385
```

```
                        0D90  3387          .SBTTL  GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
                        0D90  3388 ;++
                        0D90  3389 ; GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
                        0D90  3390 ;
                        0D90  3391 ; Functional description:
                        0D90  3392 ;
                        0D90  3393 ; This routine saves the P2 buffer for later use by the driver.
                        0D90  3394 ; The P2 buffer is saved by allocating the appropriate amount of memory from
                        0D90  3395 ; non-paged pool.   The user's quota is checked before the allocation is made.
                        0D90  3396 ; And the non-paged pool buffer is charged against the user's quota. The P2
                        0D90  3397 ; system buffer address is passed in IRP$L_SVAPTE of the IRP.
                        0D90  3398 ;
                        0D90  3399 ;
                        0D90  3400 ; Inputs:
                        0D90  3401 ;
                        0D90  3402 ;       R3 = IRP address
                        0D90  3403 ;       R4 = PCB address
                        0D90  3404 ;       R5 = UCB address
                        0D90  3405 ;
                        0D90  3406 ; Outputs:
                        0D90  3407 ;
                        0D90  3408 ;       R0 = status of buffers
                        0D90  3409 ;
                        0D90  3410 ;       R3-R5 are preserved.
                        0D90  3411 ;
                        0D90  3412 ;--
                        0D90  3413
                        0D90  3414 GET_CHAR_BUF:                                    ; Get characteristics buffer
                        0D90  3415 ;
                        0D90  3416 ; Check access to P2 buffer and check process's buffer quota
                        0D90  3417 ;
      51    04 AC     D0  0D90  3418 10$:     MOVL    P2(AP),R1                     ; Get address P2 char buf desc
                33     13  0D94  3419         BEQL    40$                           ; Br if no P2 buffer
                53     DD  0D96  3420         PUSHL   R3                            ; Save R3
 00000000'GF    16     FB  0D98  3421         JSB     G^EXE$PROBER_DSC              ; Check access to buffer
        0E      50     E9  0D9E  3422         BLBC    R0,15$                        ; Br if error
      51        51     3C  0DA1  3423         MOVZWL  R1,R1                         ; Get the length as a word
                52     DD  0DA4  3424         PUSHL   R2                            ; Save R2
 00000000'GF    16     FB  0DA6  3425         JSB     G^EXE$BUFQUOPRC               ; Check for buffered quota
                52  8ED0  0DAC  3426         POPL    R2                            ; Restore R2
                53  8ED0  0DAF  3427 15$:     POPL    R3                            ; Restore R3
          01    50     E8  0DB2  3428         BLBS    R0,30$                        ; Branch if quota ok
                05     05  0DB5  3429 20$:     RSB                                   ; Return
                           0DB6  3430
                           0DB6  3431 ;
                           0DB6  3432 ; Quota OKAY, allocate buffer and copy info.
                           0DB6  3433 ;
            0050   30     BB  0DB6  3434 30$:     BSBW    ALLOC_P2BUF                   ; Allocate buffer
          10 50    E9     BB  0DB9  3435         BLBC    R0,50$                        ; Br if error
      50  2C A3    D0     BB  0DBC  3436         MOVL    IRP$L_SVAPTE(R3),R0           ; Get P2 buffer address
                38     BB  0DC0  3437         PUSHR   #^M<R3,R4,R5>                 ; Save sacred registers
OC A0   62      51     28  0DC2  3438         MOVC3   R1,(R2),P2B_T_DATA(R0)        ; Save P2 char buffer
                38     BA  0DC7  3439         POPR    #^M<R3,R4,R5>                 ; Restore registers
          50    01     9A  0DC9  3440 40$:     MOVZBL  S^#SS$_NORMAL,R0              ; Set success
                05     05  0DCC  3441 50$:     RSB                                   ; Return
```

```
                     ODCD  3443          .SBTTL  CHECK_BUFS - CHECK P1 AND P2 BUFFERS FOR WRITE ACCESS
                     ODCD  3444 ;++
                     ODCD  3445 ; CHECK_BUFS - CHECK P1 AND P2 BUFFERS FOR WRITE ACCESS
                     ODCD  3446 ;
                     ODCD  3447 ; Functional description:
                     ODCD  3448 ;
                     ODCD  3449 ; This routines checks the P1 and P2 buffers for write access if supplied.
                     ODCD  3450 ;
                     ODCD  3451 ; Inputs:
                     ODCD  3452 ;
                     ODCD  3453 ;       R1 = Size of P1 buffer needed for write access
                     ODCD  3454 ;       R3 = IRP address
                     ODCD  3455 ;       R4 = PCB address
                     ODCD  3456 ;       R5 = UCB address
                     ODCD  3457 ;       R7 = Function code
                     ODCD  3458 ;
                     ODCD  3459 ; Outputs:
                     ODCD  3460 ;
                     ODCD  3461 ;       R0 is destroyed.
                     ODCD  3462 ;       R1 = Length of P2 buffer (zero if no P2 buffer)
                     ODCD  3463 ;       R2 = Address of P2 buffer in user's process space
                     ODCD  3464 ;
                     ODCD  3465 ;
                     ODCD  3466 ;       No RETURN on NO ACCESS
                     ODCD  3467 ;
                     ODCD  3468 ; Implicit Outputs:
                     ODCD  3469 ;
                     ODCD  3470 ;       IRP$V_FUNC bit set in IRP$W_STS by EXE$READCHK subroutine.
                     ODCD  3471 ;
                     ODCD  3472 ;--
                     ODCD  3473
                     ODCD  3474 CHECK_BUFS:
          27   10     ODCD  3475          BSBB    CHECK_P1                        ; Check P1 buffer
                     ODCF  3476 CHECK_P2:
          51   D4     ODCF  3477          CLRL    R1                              ; Assume no P2 buffer desc
    52    04   AC     D0    ODD1  3478    MOVL    P2(AP),R2                       ; Get address of P2 desc
          18   13     ODD5  3479          BEQL    10$                             ; Br if no P2
                     ODD7  3480          IFNORD  #8,(R2),ACCESS                  ; Br if no access
       51   62   3C   ODDD  3481          MOVZWL  (R2),R1                         ; Get length of buffer
          0D   13     ODE0  3482          BEQL    10$                             ; Br if zero
    50    04   A2     D0    ODE2  3483    MOVL    DSC$A_POINTER(R2),R0            ; Get buffer address
00000000'GF   16     ODE6  3484          JSB     G^EXE$READCHK                   ; Check write access to buffer
                     ODEC  3485                                                  ; (no return no access)
                     ODEC  3486                                                  ; Also sets IRP$V_FUNC in IRP
       52   50   D0   ODEC  3487          MOVL    R0,R2                           ; Copy buffer address
                     05    ODEF  3488 10$: RSB                                   ; Return to caller
                     ODF0  3489
       50   0C   9A   ODF0  3490 ACCESS: MOVZBL  S^#SS$_ACCVIO,R0                ; Return access violation
            F550   31   ODF3  3491          BRW     ABORTIO                         ; Abort the I/O request
```

```
                     0DF6  3493          .SBTTL  CHECK_P1 - CHECK P1 BUFFER ADDRESS FOR WRITE ACCESS
                     0DF6  3494 ;++
                     0DF6  3495 ; CHECK_P1 - CHECK P1 BUFFER ADDRESS FOR WRITE ACCESS
                     0DF6  3496 ;
                     0DF6  3497 ; Functional description:
                     0DF6  3498 ;
                     0DF6  3499 ; This routine checks the P1 buffer and if okay, the buffer address
                     0DF6  3500 ; is saved in IRP$L_MEDIA of the IRP.
                     0DF6  3501 ;
                     0DF6  3502 ; Inputs:
                     0DF6  3503 ;
                     0DF6  3504 ;       R1 = Size of buffer for write access
                     0DF6  3505 ;       R3 = IRP address
                     0DF6  3506 ;       R4 = PCB address
                     0DF6  3507 ;       R5 = UCB address
                     0DF6  3508 ;       R7 = Function code
                     0DF6  3509 ;
                     0DF6  3510 ; Outputs:
                     0DF6  3511 ;
                     0DF6  3512 ;       R0 is destroyed.
                     0DF6  3513 ;
                     0DF6  3514 ;       No RETURN on NO ACCESS.
                     0DF6  3515 ;
                     0DF6  3516 ; Implicit Outputs:
                     0DF6  3517 ;
                     0DF6  3518 ;       IRP$L_MEDIA(R3) = User P1 buffer address.
                     0DF6  3519 ;       IRP$V_FUNC bit set in IRP$W_STS by EXE$READCHK subroutine.
                     0DF6  3520 ;
                     0DF6  3521 ;--
                     0DF6  3522
                     0DF6  3523 CHECK_P1:
        38 A3   D4   0DF6  3524          CLRL    IRP$L_MEDIA(R3)                 ; Assume no P1 buffer
     50    6C   D0   0DF9  3525          MOVL    P1(AP),R0                       ; Get address of user buffer
           06   13   0DFC  3526          BEQL    10$                             ; Br if none
00000000'GF    16   0DFE  3527          JSB     G^EXE$READCHK                   ; Check access to buffer
                    0E04  3528                                                  ; (No return - no access)
 3C A3    50   D0   0E04  3529 10$:     MOVL    R0,IRP$L_XQ_USERBUF(R3)         ; Save P1 buffer address in IRP
                    05   0E08  3530          RSB                             ; Return to caller
```

```
                                0E09  3532          .SBTTL  ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
                                0E09  3533 ;++
                                0E09  3534 ; ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
                                0E09  3535 ;
                                0E09  3536 ; Functional description:
                                0E09  3537 ;
                                0E09  3538 ; This routine allocates a system buffer and returns the address in the IRP at
                                0E09  3539 ; IRP$L_SVAPTE.  The size of the allocation, including buffer header must
                                0E09  3540 ; be at least 24 bytes in length.
                                0E09  3541 ;
                                0E09  3542 ; Inputs:
                                0E09  3543 ;
                                0E09  3544 ;       R1 = Size of allocation desired
                                0E09  3545 ;       R3 = IRP address
                                0E09  3546 ;
                                0E09  3547 ; Outputs:
                                0E09  3548 ;
                                0E09  3549 ;       R0 = status of request
                                0E09  3550 ;
                                0E09  3551 ;       R1-R5 are preserved.
                                0E09  3552 ;
                                0E09  3553 ; Implicit Outputs:
                                0E09  3554 ;
                                0E09  3555 ;       IRP$L_SVAPTE(R3) = address of system buffer
                                0E09  3556 ;       IRP$W_BOFF(R3) = byte count charged to user's process
                                0E09  3557 ;       IRP$W_BCNT(R3) = original byte count requested
                                0E09  3558 ;
                                0E09  3559 ;       All parts of the P2 buffer header are initialized, except for the
                                0E09  3560 ;       user's P2 buffer address.
                                0E09  3561 ;
                                0E09  3562 ;--
                                0E09  3563
                                0E09  3564 ALLOC_P2BUF:                                    ; Allocate a non-paged buffer
          51    D5              0E09  3565          TSTL    R1                            ; Zero length buffer?
          4D    13              0E0B  3566          BEQL    30$                           ; Br if yes
          0E    BB              0E0D  3567          PUSHR   #^M<R1,R2,R3>                 ; Save registers
    32 A3 51    B0              0E0F  3568          MOVW    R1,IRP$W_BCNT(R3)             ; Save original byte count
       0C 51    D1              0E13  3569          CMPL    R1,S^#24=P2B_C_LENGTH         ; Is buffer big enough?
          03    1A              0E16  3570          BGTRU   5$                            ; Br if yes
       51 0C    D0              0E18  3571          MOVL    S^#24-P2B_C_LENGTH,R1         ; Else, set size to minimum
       51 0C    C0              0E1B  3572 5$:      ADDL2   S^#P2B_C_LENGTH,R1            ; Add in size of header
00000000'GF     16              0E1E  3573          JSB     G^EXE$BUFQUOPRC               ; Check for buffered quota
       0E 50    E9              0E24  3574          BLBC    R0,10$                        ; Branch if quota bad
                                0E27  3575 ;
                                0E27  3576 ; Quota OKAY, allocate buffer and copy info.
                                0E27  3577 ;
          51    DD              0E27  3578          PUSHL   R1                            ; Save size to charge user
00000000'GF     16              0E29  3579          JSB     G^EXE$ALLOCBUF                ; Go allocate a buffer
       06 50    E8              0E2F  3580          BLBS    R0,20$                        ; Br if success
       5E 04    C0              0E32  3581          ADDL    #4,SP                         ; Pop saved size
          0E    BA              0E35  3582 10$:     POPR    #^M<R1,R2,R3>                 ; Restore registers
                05              0E37  3583          RSB                                   ; Return with error code in R0
                                0E38  3584 ;
                                0E38  3585 ; System buffer allocated decrement user's quota
                                0E38  3586 ;
          53 8ED0               0E38  3587 20$:     POPL    R3                            ; Restore user quota charge
    62    0C A2    9E           0E3B  3588          MOVAB   P2B_T_DATA(R2),P2B_L_POINTER(R2) ; Set address to start of data
```

```
                08 A2    53   B0  0E3F  3589          MOVW    R3,P2B_W_SIZE(R2)                ; Save buffer size in buffer
                50       52   D0  0E43  3590          MOVL    R2,R0                            ; Save P2 char buf addr
             52 0080 C4       D0  0E46  3591          MOVL    PCB$L_JIB(R4),R2                 ; Get JIB address
                20 A2    53   C2  0E4B  3592          SUBL    R3,JIB$L_BYTCNT(R2)              ; Decrement user's quota
                         0E   BA  0E4F  3593          POPR    #^M<R1,R2,R3>                    ; Restore registers
                2C A3    50   D0  0E51  3594          MOVL    R0,IRP$L_SVAPTE(R3)              ; Save P2 buffer address in IRP
             30 A3    08 A0   B0  0E55  3595          MOVW    P2B_W_SIZE(R0),IRP$W_BOFF(R3)    ; Return buffer size in IRP
                50       01   9A  0E5A  3596 30$:      MOVZBL  S^#SS$_NORMAL,R0                 ; Set success
                              05  0E5D  3597          RSB                                      ; Return to caller
```

```
                     0E5E  3599          .SBTTL  STARTIO - START I/O OPERATION
                     0E5E  3600 ;++
                     0E5E  3601 ; STARTIO - START I/O OPERATION
                     0E5E  3602 ;
                     0E5E  3603 ; Functional description:
                     0E5E  3604 ;
                     0E5E  3605 ; This routine is called when an IRP is ready to be processed by the driver.
                     0E5E  3606 ; The request is dispatched to the appropriate routine base on the internal
                     0E5E  3607 ; function code in the IRP.
                     0E5E  3608 ;
                     0E5E  3609 ; Inputs:
                     0E5E  3610 ;
                     0E5E  3611 ;       R3 = IRP address
                     0E5E  3612 ;       R5 = UCB address
                     0E5E  3613 ;
                     0E5E  3614 ;       IPL = FIPL
                     0E5E  3615 ;
                     0E5E  3616 ; Outputs:
                     0E5E  3617 ;
                     0E5E  3618 ;       R0-R2,R4 are destroyed.
                     0E5E  3619 ;--
                     0E5E  3620
                     0E5E  3621 STARTIO::                                ; Process an I/O packet
54   24 A5   D0      0E5E  3622          MOVL    UCB$L_CRB(R5),R4       ; Get CRB address
54   10 A4   D0      0E62  3623          MOVL    CRB$L_AUXSTRUC(R4),R4  ; Get CDB address
51   21 A3   9A      0E66  3624          MOVZBL  IRP$B_XQ_FUNC(R3),R1   ; Get the internal function code
                     0E6A  3625 10$:     $DISPATCH       R1,TYPE=B,-
                     0E6A  3626          <-      ;function       action
                     0E6A  3627
                     0E6A  3628                  <XQ_FC_V_INIT   STARTUP>,-      ; Startup request
                     0E6A  3629                  <XQ_FC_V_STOP   SHUT>,-         ; Shutdown request
                     0E6A  3630                  <XQ_FC_V_CHMODE CHMODE>,-       ; Set new multicast list
                     0E6A  3631          >
                     0E7C  3632 ;
                     0E7C  3633 ; Other request type
                     0E7C  3634 ;
                     0E7C  3635          BUG_CHECK NOBUFPCKT,FATAL      ; Fatal error
                     0E80  3636
                     0E80  3637 ;
                     0E80  3638 ; Startup unit's protocol
                     0E80  3639 ;
                     0E80  3640 STARTUP:                                 ; Startup unit's protocol
   4C   10           0E80  3641          BSBB    START                  ; Start protocol
01 50   E9           0E82  3642          BLBC    R0,10$                 ; Br if error on startup
         05          0E85  3643          RSB                            ; Else, return to caller
                     0E86  3644
   50   DD           0E86  3645 10$:     PUSHL   R0                     ; Save error return
   2A   10           0E88  3646          BSBB    STOP                   ; Shutdown unit
   50 8ED0           0E8A  3647          POPL    R0                     ; Restore error return
 0D4C   31           0E8D  3648          BRW     IO_DONE                ; Complete the I/O request
                     0E90  3649
                     0E90  3650 ;
                     0E90  3651 ; Shutdown UNIT's protocol
                     0E90  3652 ;
                     0E90  3653 SHUT:                                    ; Shutdown protocol
0214 C4   55   D1    0E90  3654          CMPL    R5,CDB_L_PRMUSER(R4)   ; Are we the PROMISCUOUS user?
          14   12    0E95  3655          BNEQ    10$                    ; Br if not
```

```
                0214 C4    D4  0E97  3656          CLRL    CDB_L_PRMUSER(R4)       ; Else, clear the PROMISCUOUS user
                   01      90  0E9B  3657          MOVB    #NMA$C_STATE_OFF,-      ; Don't forget about the CDB
                024B C4        0E9D  3658                  CDB_B_PRM(R4)           ;  parameter
                   04CC    30  0EA0  3659          BSBW    SETUP_MODE              ; Get setup buffer
                 0D 50     E9  0EA3  3660          BLBC    R0,90$                  ; Br if error
                   03      90  0EA6  3661          MOVB    #XQ_FC_V_STOP,-         ; Set function request
                20 A2          0EA8  3662                  CXB$B_XQ_FUNC(R2)       ;
                           05  0EAA  3663          RSB                             ; Return to complete function
                               0EAB  3664
                   07      10  0EAB  3665  10$:    BSBB    STOP                    ; Shutdown unit
                50    01      9A  0EAD  3666          MOVZBL  S^#SS$_NORMAL,R0        ; Return success
                   0D29    31  0EB0  3667          BRW     IO_DONE                 ; Complete I/O request
                               0EB3  3668
                           05  0EB3  3669  90$:     RSB                             ; Return to caller
                               0EB4  3670  ;
                               0EB4  3671  ; STOP the unit
                               0EB4  3672  ;
                               0EB4  3673  STOP:                                   ; Stop the protocol
                54   24 A5     D0  0EB4  3674          MOVL    UCB$L_CRB(R5),R4        ; Get CRB address
                54   10 A4     D0  0EB8  3675          MOVL    CRB$L_AUXSTRUC(R4),R4   ; Get CDB address
                     122C      31  0EBC  3676          BRW     SHUTDOWN_PROTYP         ; Shutdown the unit
                               0EBF  3677  ;
                               0EBF  3678  ; CHMODE - set new multicast list
                               0EBF  3679  ;
                               0EBF  3680  CHMODE:
                   04AD    30  0EBF  3681          BSBW    SETUP_MODE              ; Get XMIT setup buffer
                   06 50   E9  0EC2  3682          BLBC    R0,10$                  ; Exit if error
                   06      90  0EC5  3683          MOVB    #XQ_FC_V_CHMODE,-       ; Set function request
                20 A2          0EC7  3684                  CXB$B_XQ_FUNC(R2)       ;
                   9E      17  0EC9  3685          JMP     @(SP)+                  ; Call back caller and return
                               0ECB  3686
                   0D0E    31  0ECB  3687  10$:    BRW     IO_DONE                 ; Complete I/O request
```

```
                        .SBTTL  START - START UNIT'S PROTOCOL
                ;++
                ; START - START UNIT'S PROTOCOL
                ;
                ; Functional description:
                ;
                ; This routine initiates the protocol on the unit. The QNA is reset if first
                ; unit online.  The free list is filled and the first receive started.
                ; If a failure occurs the unit shutdown sequence is entered.
                ;
                ; Inputs:
                ;
                ;       R3 = IRP address
                ;       R5 = UCB address
                ;
                ;       IPL = FIPL
                ;
                ; Implicit inputs:
                ;
                ;       IRP$L_MEDIA contains a copy of the mode buffer specified by the user.
                ;       IRP$W_BOFF contains the quota taken from the user for the unit.
                ;
                ; Outputs:
                ;
                ;       R0 = Status return for startup request.
                ;       R1,R2,R4 are destroyed.
                ;       R3,R5 are preserved.
                ;
                ;--

                START::                                 ; Start protocol operation
00 91           CMPB    #NMA$C_LINPR_POI,-              ;% Are we in PT-TO-PT mode?
00D8 C5                 UCB$B_XQ_PRO(R5)                ;%
   24 12        BNEQ    3$                              ;% Br if not
51 00C4 8F 3C   MOVZWL  #IRP$C_LENGTH,R1                ;% Set size of an IRP
   53 DD        PUSHL   R3                              ;% Save R3
00000000'GF 16  JSB     G^EXE$ALONONPAGED               ;% Allocate the IRP
   53 8ED0      POPL    R3                              ;% Restore R3
   06 50 E8     BLBS    R0,1$                           ;% Br if success
50 0124 8F 3C   MOVZWL  #SS$_INSFMEM,R0                 ;% Set error return
   05           RSB                                     ;% Return to caller

0191 C5 52 D0   1$:     MOVL    R2,UCB$L_XQ_STIRP(R5)   ;% Save startup IRP
0060 8F A8      BISW    #UCB$M_XQ_START!UCB$M_XQ_STACK,- ;% We are now in the startup
  68 A5                 UCB$W_DEVSTS(R5)                ;%   and stack wait state

00BC C5 D5      3$:     TSTL    UCB$L_XQ_CPID(R5)       ; Creator PID saved already?
   06 12        BNEQ    5$                              ; Br if yes
00BC C5 20 A5 D0        MOVL    UCB$L_CPID(R5),UCB$L_XQ_CPID(R5) ; Else, save creator PID
                ;
                ; Set up idle UCB
                ;
0800 8F 3C      5$:     MOVZWL  #XM$M_STS_ACTIVE,-
  44 A5                 UCB$L_DEVDEPEND(R5)             ; Reset status and error summary
21 A3 05 91     CMPB    #XQ_FC_V_RESTART,IRP$B_XQ_FUNC(R3) ; Is this a re-start operation?
   06 13        BEQL    8$                              ; Br if yes - don't reset the PID
00B8 C5 0C A3 D0        MOVL    IRP$L_PID(R3),UCB$L_XQ_PID(R5) ; Save starter's PID
```

```
                        0F17  3746 ;
                        0F17  3747 ; Check for CDB
                        0F17  3748 ;
      54    24 A5   D0  0F17  3749 8$:     MOVL    UCB$L_CRB(R5),R4          ; Get CRB address
      52    10 A4   D0  0F1B  3750         MOVL    CRB$L_AUXSTRUC(R4),R2     ; Get CDB address, crash if not present
         020F C2    96  0F1F  3751         INCB    CDB_B_UNTCNT(R2)          ; One more unit on this controller
               00   E0  0F23  3752         BBS     #CDB_STS_V_INITED,-       ; Br if already inited
      03 024A C2        0F25  3753                 CDB_B_STS(R2),10$         ;
            0093   31  0F29  3754         BRW     20$                       ; Else, init CDB
                        0F2C  3755
         54    52   D0  0F2C  3756 10$:    MOVL    R2,R4                     ; Copy CDB address
   50    00DD C5   9E  0F2F  3757         MOVAB   UCB$B_XQ_CDBPRM(R5),R0    ; Get UCB parameter address
   51    024D C4   9E  0F34  3758         MOVAB   CDB_B_SETPRM(R4),R1       ; Get CDB parameter address
         52    01   9A  0F39  3759         MOVZBL  #UCB$C_XQ_CDBPRM,R2       ; Set size of parameter list
                        0F3C  3760
                        0F3C  3761 ;
                        0F3C  3762 ; Check order of UCB parameters
                        0F3C  3763 ;
                        0F3C  3764         ASSUME  UCB$B_XQ_CON EQ UCB$B_XQ_CDBPRM
                        0F3C  3765 ;
                        0F3C  3766 ; Check order of CDB parameters
                        0F3C  3767 ;
                        0F3C  3768         ASSUME  CDB_B_CON EQ CDB_B_SETPRM
                        0F3C  3769
                        0F3C  3770         ASSUME  NMA$C_STATE_ON NE -1
                        0F3C  3771         ASSUME  NMA$C_STATE_OFF NE -1
         61    60   91  0F3C  3772 13$:    CMPB    (R0),(R1)                 ; Match?
               73   12  0F3F  3773         BNEQ    18$                       ; Br if no
         80    81   90  0F41  3774         MOVB    (R1)+,(R0)+               ; Store CDB value in UCB
            F5 52   F5  0F44  3775         SOBGTR  R2,13$                    ; Loop if more to check
                        0F47  3776 ;
                        0F47  3777 ; NOW, check if user has given a hardware physical address.
                        0F47  3778 ;
                        0F47  3779 ;       R0 = Address of parameters in UCB
                        0F47  3780 ;       R1 = Address of parameters in CDB
                        0F47  3781 ;
                        0F47  3782         ASSUME  CDB_G_PHA EQ CDB_B_CON+1
                        0F47  3783         ASSUME  UCB$G_XQ_PHA EQ UCB$B_XQ_CON+1
60    FFFFFFFF 8F   D1  0F47  3784         CMPL    #-1,(R0)                  ; Is user physical address defined?
               08   12  0F4E  3785         BNEQ    15$                       ; Br if yes
 04 A0    FFFF 8F   B1  0F50  3786         CMPW    #-1,4(R0)                 ; Is user physical address defined?
               10   13  0F56  3787         BEQL    16$                       ; Br if not
 40 A3    0B04 8F   B0  0F58  3788 15$:    MOVW    #NMA$C_PCLI_PHA,IRP$W_XQ_CODE(R3) ; Assume bad physical address
         81    80   D1  0F5E  3789         CMPL    (R0)+,(R1)+               ; Physical address match??
               58   12  0F61  3790         BNEQ    19$                       ; Br if no
         81    80   B1  0F63  3791         CMPW    (R0)+,(R1)+               ; Still match??
               53   12  0F66  3792         BNEQ    19$                       ; Br if no
         024E C4   D0  0F68  3793 16$:    MOVL    CDB_G_PHA(R4),-           ; Return hardware set address
         00DE C5         0F6C  3794                 UCB$G_XQ_PHA(R5)          ;  just in case we defaulted
         0252 C4   B0  0F6F  3795         MOVW    CDB_G_PHA+4(R4),-         ;       ...
         00E2 C5         0F73  3796                 UCB$G_XQ_PHA+4(R5)        ;       ...
                        0F76  3797 ;
                        0F76  3798 ; Check users buffer size - must not be more than twice the hardware buffer
                        0F76  3799 ; size. (Already has been checked against max message size).
                        0F76  3800 ;
 40 A3    0AF1 8F   B0  0F76  3801         MOVW    #NMA$C_PCLI_BUS,IRP$W_XQ_CODE(R3) ; Assume bad buffer size
      51    0110 C4   3C  0F7C  3802         MOVZWL  CDB_W_BSZ(R4),R1          ; Get device buffer size
```

```
                05 00DC C5    E8  0F81  3803          BLBS    UCB$B_XQ_DCH(R5),17$     ; Br if user can't do data chaining
                              0F86  3804                                             ; && Maybe this is an Internal IRP user
              51   0110 C4    A0  0F86  3805          ADDW    CDB_W_BSZ(R4),R1         ; Compute twice the normal buffer size
                51    42 A5    B1  0F8B  3806 17$:     CMPW    UCB$W_DEVBUFSIZ(R5),R1   ; Is buffer size okay?
                        2A     1A  0F8F  3807          BGTRU   19$                      ; Br if too large
                     40 A3     B4  0F91  3808          CLRW    IRP$W_XQ_CODE(R3)        ; No bad parameters
                        10     A8  0F94  3809          BISW    #UCB$M_XQ_RUN,-          ; Indicate we have entered RUN mode
                     68 A5         0F96  3810                  UCB$W_DEVSTS(R5)         ;
                      178D     30  0F98  3811          BSBW    MOVE_MULTI               ; Copy multicast address list
                               0F9B  3812          ASSUME  NMA$C_STATE_ON EQ 0
                               0F9B  3813          ASSUME  NMA$C_STATE_OFF EQ 1
             07 00DA C5    E8  0F9B  3814          BLBS    UCB$B_XQ_PRM(R5),173$    ; Br if not promiscuous
                00DA C5    90  0FA0  3815          MOVB    UCB$B_XQ_PRM(R5),-       ; Else, enable promiscuous mode
                024B C4        0FA4  3816                  CDB_B_PRM(R4)
                   03C5    30  0FA7  3817 173$:    BSBW    SETUP_MODE               ; Allocate setup mode buffer
                     06 50 E9  0FAA  3818          BLBC    R0,175$                  ; Exit if error
                        06     90  0FAD  3819          MOVB    #XQ_FC_V_CHMODE,-        ; Set function request
                     20 A2         0FAF  3820                  CXB$B_XQ_FUNC(R2)        ;
                        9E     16  0FB1  3821          JSB     @(SP)+                   ; Complete request
                               05  0FB3  3822 175$:    RSB                              ; Return to caller
                               0FB4  3823 ;
                               0FB4  3824 ; Error on parameter validation
                               0FB4  3825 ;
          40 A3   F1EB CF42    B0  0FB4  3826 18$:     MOVW    BAD_PARAM_TBL-2[R2],IRP$W_XQ_CODE(R3) ; Return parameter code
                    50   14    9A  0FBB  3827 19$:     MOVZBL  S^#SS$_BADPARAM,R0       ; Return bad parameter error
                               05  0FBE  3828          RSB                              ; Return to caller
                               0FBF  3829 ;
                               0FBF  3830 ; Initialize CDB
                               0FBF  3831 ;
                51   020E 8F    3C  0FBF  3832 20$:     MOVZWL  #CDB_C_ZERO,R1           ; Get portion of CDB to init with zero
                          3E   BB  0FC4  3833          PUSHR   #^M<R1,R2,R3,R4,R5>      ; Save registers
      62   51   00   62    00    2C  0FC6  3834          MOVC5   #0,(R2),#0,R1,(R2)       ; Zero the structure
                          3E   BA  0FCC  3835          POPR    #^M<R1,R2,R3,R4,R5>      ; Restore registers
                               0FCE  3836          ASSUME  CDB_L_FQFL EQ 0
                               0FCE  3837          ASSUME  CDB_L_FQBL EQ CDB_L_FQFL+4
                     54   82    7E  0FCE  3838          MOVAQ   (R2)+,R4                 ; Skip link pointers, copy CDB address
                               0FD1  3839          ASSUME  CDB_W_SIZE EQ CDB_L_FQBL+4
                     82   51    B0  0FD1  3840          MOVW    R1,(R2)+                 ; Store size of structure
                               0FD4  3841          ASSUME  CDB_B_TYPE EQ CDB_W_SIZE+2
                               0FD4  3842          ASSUME  CDB_B_FIPL EQ CDB_B_TYPE+1
                82   0833 8F    B0  0FD4  3843          MOVW    #<IPL$_XQ_FIPL@8>!DYN$C_CDB,(R2)+ ; Set structure type and FIPL
                               0FD9  3844          ASSUME  CDB_L_FPC EQ CDB_B_FIPL+1
           82   000016E0'EF    9E  0FD9  3845          MOVAB   FORK_PROC,(R2)+          ; Set fork process address
                               0FE0  3846          ASSUME  CDB_L_FR3 EQ CDB_L_FPC+4
                               0FE0  3847          ASSUME  CDB_L_FR4 EQ CDB_L_FR3+4
                     82    7C  0FE0  3848          CLRQ    (R2)+                    ; Clear fork R3 and R4
                               0FE2  3849          ASSUME  CDB_B_NEXTXMT EQ CDB_L_FR4+4
                               0FE2  3850          ASSUME  CDB_B_NEXTRCV EQ CDB_B_NEXTXMT+1
                               0FE2  3851          ASSUME  CDB_B_RCVMAP EQ CDB_B_NEXTRCV+1
                               0FE2  3852          ASSUME  CDB_B_XMTMAP EQ CDB_B_RCVMAP+1
                     82    D4  0FE2  3853          CLRL    (R2)+                    ; Clear slot in use flags
                               0FE4  3854          ASSUME  CDB_L_RCVMAP EQ CDB_B_XMTMAP+1
                               0FE4  3855          ASSUME  CDB_L_XMTMAP EQ CDB_L_RCVMAP+<4*<MAX_C_RCV-1>>
                51   0A    9A  0FE4  3856          MOVZBL  #<MAX_C_XMT-1>+<MAX_C_RCV-1>,R1 ; Set number of mapping vectors
                82   01    CE  0FE7  3857 30$:     MNEGL   #1,(R2)+                 ; Indicate no mapping info
                  FA 51    F5  0FEA  3858          SOBGTR  R1,30$                   ; Loop if more
                               0FED  3859          ASSUME  CDB_L_RRINGPA EQ CDB_L_XMTMAP+<4*<MAX_C_XMT-1>>
```

```
                52    00000098 8F   C0   0FED  3860          ADDL    #CDB_Q_QUEUES-CDB_L_RRINGPA,R2  ; Skip ring entry pointers
                      51    06 9A   0FF4  3861          MOVZBL  #CDB_C_QUEUES,R1        ; Set number of queue listheads
                      82    62 DE   0FF7  3862 40$:     MOVAL   (R2),(R2)+              ; Set forward link
                    82    FC A2 D0   0FFA  3863          MOVL    -4(R2),(R2)+            ; Set backward link
                      F6 51 F5   0FFE  3864          SOBGTR  R1,40$                  ; Loop if more listheads
                                 1001  3865          SETBIT  #CDB_STS_V_INITED,CDB_B_STS(R4) ; Set initial status bits
                                 1007  3866 ;
                                 1007  3867 ; Initialize CDB defaults
                                 1007  3868 ;
                      00D6 C5    B0   1007  3869          MOVW    UCB$W_XQ_BSZ(R5),-      ; Init CDB buffer size
                      0110 C4         100B  3870                  CDB_W_BSZ(R4)           ;
            0112 C4   00D2 C5    B0   100E  3871          MOVW    UCB$W_XQ_HBQ(R5),CDB_W_QUOTA(R4) ; Set initial quota
                                 1015  3872
                                 1015  3873          ASSUME  INIT_C_AQUOTA LE 255
                                 1015  3874          ASSUME  CDB_B_MQUOTA EQ CDB_B_AQUOTA+1
                      0200 8F    B0   1015  3875          MOVW    #<INIT_C_AQUOTA@8>,-    ; Initialize Maximum QUOTA
                      02F2 C4         1019  3876                  CDB_B_AQUOTA(R4)        ; and zero Additional QUOTA
                                 101C  3877          ASSUME  UCB$B_XQ_MLT EQ UCB$B_XQ_PRM+1
                                 101C  3878          ASSUME  CDB_B_MLT EQ CDB_B_PRM+1
            024B C4   00DA C5    B0   101C  3879          MOVW    UCB$B_XQ_PRM(R5),CDB_B_PRM(R4) ; Set the promiscuous mode
                                 1023  3880                                          ;  and the all multicast enable
                50    00DD C5   9E   1023  3881          MOVAB   UCB$B_XQ_CDBPRM(R5),R0  ; Get address of UCB parameters
                51    024D C4   9E   1028  3882          MOVAB   CDB_B_SETPRM(R4),R1     ; Get address of CDB parameters
                      52    07  9A   102D  3883          MOVZBL  #UCB$C_XQ_SETPRM,R2     ; Set size of parameter list
                      81    80  90   1030  3884 45$:     MOVB    (R0)+,(R1)+             ; Store parameters
                        FA 52   F5   1033  3885          SOBGTR  R2,45$                  ; Loop if more
                          16EF  30   1036  3886          BSBW    MOVE_MULTI              ; Copy multicast address list
                      0110 C4   B0   1039  3887          MOVW    CDB_W_BSZ(R4),-         ; Set buffer size
                        7E A5         103D  3888                  UCB$W_BCNT(R5)
                                 103F  3889          PUSHQ   R6                      ; Save R6,R7
                          56    54  D0   1042  3890          MOVL    R4,R6                   ; Save CDB address
                        54    24 A5   D0   1045  3891          MOVL    UCB$L_CRB(R5),R4        ; Get CRB address
                        57    28 A5   D0   1049  3892          MOVL    UCB$L_DDB(R5),R7        ; Get DDB address
            0118 C6     04 A7   D0   104D  3893          MOVL    DDB$L_UCB(R7),CDB_L_UCB0(R6) ; Set UCB #0 address
                                 1053  3894
                                 1053  3895          CPUDISP <<790,50$>,-
                                 1053  3896                  <780,50$>,-
                                 1053  3897                  <750,50$>,-
                                 1053  3898                  <730,50$>,-
                                 1053  3899                  <UV1,70$>>
                                 106D  3900 50$:     ;
                                 106D  3901          ; Allocate map registers for receive buffers and one transmit buffer.
                                 106D  3902          ;
            7C A5     01FF 8F    B0   106D  3903          MOVW    #511,UCB$W_BOFF(R5)     ; Set worst case byte offset
                                 1073  3904          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                                 1073  3905          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                      34 A4     D4   1073  3906          CLRL    CRB$L_INTD+VEC$W_MAPREG(R4) ; Clear map register + datapath
                        56    1C A6   DE   1076  3907          MOVAL   CDB_L_RCVMAP(R6),R6     ; Get mapping slot address
                          57    07  9A   107A  3908          MOVZBL  #MAX_T_RCV-1,R7         ; Get number of receive slots
                    00000000'GF     16   107D  3909 55$:     JSB     G^IOC$ALOUBAMAP         ; Allocate a set of map registers
                      39 50     E9   1083  3910          BLBC    R0,60$                  ; Br if unavailable
                      86    34 A4   D0   1086  3911          MOVL    CRB$L_INTD+VEC$W_MAPREG(R4),(R6)+ ; Save map info
                        F0 57   F5   108A  3912          SOBGTR  R7,55$                  ; Continue
                                 108D  3913
                        05EE 8F    3C   108D  3914          MOVZWL  #MAX_PKT_SIZE+18,-      ; Set transmit buffer size to max
                          7E A5         1091  3915                  UCB$W_BCNT(R5)          ;  Ethernet packet size + header
                    00000000'GF     16   1093  3916          JSB     G^IOC$ALOUBAMAP         ; Allocate a set of transmit registers
```

```
                23 50    E9  1099  3917          BLBC    R0,60$                          ; Br if unavailable
                                1099  3918          ASSUME  CDB_L_XMTMAP EQ CDB_L_RCVMAP+<4*<MAX_C_RCV-1>>
              66   34 A4    D0  1099  3919          MOVL    CRB$L_INTD+VEC$W_MAPREG(R4),(R6) ; Save transmit map info
                                10A0  3920          ;
                                10A0  3921          ; Allocate mapping for QNA RING structures.
                                10A0  3922          ;
              56   10 A4    D0  10A0  3923          MOVL    CRB$L_AUXSTRUC(R4),R6   ; Get CDB address
                                10A4  3924
                                10A4  3925          ASSUME  CDB_G_MAPPED EQ CDB_G_RRING
                                10A4  3926          ASSUME  CDB_G_XRING EQ CDB_G_RRING+RCV_K_LENGTH
           57   0166 C6    9E  10A4  3927          MOVAB   CDB_G_MAPPED(R6),R7     ; Get starting ring address
                FE00 8F    AB  10A9  3928          BICW3   #^C<VA$M_BYTE>,-        ; Get PCBB byte offset
             7C A5   57        10AD  3929          R7,UCB$W_BOFF(R5)
        7E A5   00A8 8F    B0  10B0  3930          MOVW    #CDB_C_MAPPED,UCB$W_BCNT(R5) ; Set Block size
            00000000'GF    16  10B6  3931          JSB     G^IOC$ALOUBAMAP         ; Allocate map registers
                  09 50    E8  10BC  3932          BLBS    R0,65$                  ; Br if allocated
                              10BF  3933 60$:       POPQ    R6                      ; Restore R6, R7
           50   0344 8F    3C  10C2  3934          MOVZWL  #SS$_INSFMAPREG,R0      ; Set insufficient map registers
                       05     10C7  3935          RSB                             ; Return with error
                              10C8  3936
                              10C8  3937          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                              10C8  3938          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                34 A4    D0  10C8  3939 65$:     MOVL    CRB$L_INTD+VEC$W_MAPREG(R4),- ; Save RING mapping info
                0162 C6        10CB  3940                  CDB_L_RINGMAP(R6)       ;
                              10CE  3941          ;
                              10CE  3942          ; Convert the virtual RING address to a UNIBUS mapped address
                              10CE  3943          ;
                     09    EF  10CE  3944          EXTZV   S^#VA$V_VPN,-           ; Get RING page number
                51   57   15     10D0  3945          S^#VA$S_VPN,R7,R1       ;
             50   00000000'GF    D0  10D3  3946          MOVL    G^MMG$GL_SPTBASE,R0     ; Get SPT address
                78 A5   6041    DE  10DA  3947          MOVAL   (R0)[R1],UCB$L_SVAPTE(R5) ; Set PTE address
                00000000'GF    16  10DF  3948          JSB     G^IOC$LOADUBAMAP        ; Load the PCBB map registers
                     02    07    EF  10E5  3949          EXTZV   #7,#2,-                 ; Get BA16-BA17
                  51   34 A4        10E8  3950          CRB$L_INTD+VEC$W_MAPREG(R4),R1 ;
                51   51   10    78  10EB  3951          ASHL    #16,R1,R1               ; Move to high word
                  51   7C A5    B0  10EF  3952          MOVW    UCB$W_BOFF(R5),R1       ; Set BA0-BA8
                       34 A4    F0  10F3  3953          INSV    CRB$L_INTD+VEC$W_MAPREG(R4),- ; Set BA9-BA15
                51   07   09        10F6  3954          #9,#7,R1                ;
                          22    11  10F9  3955          BRB     80$                     ; Continue
                              10FB  3956
                              10FB  3957 70$:     ;
                              10FB  3958          ; Compute physical address t0 start of rings for u-VAX I.
                              10FB  3959          ;
                              10FB  3960          ASSUME  CDB_G_MAPPED EQ CDB_G_RRING
                              10FB  3961          ASSUME  CDB_G_XRING EQ CDB_G_RRING+RCV_K_LENGTH
           57   0166 C6    9E  10FB  3962          MOVAB   CDB_G_MAPPED(R6),R7     ; Get starting ring address
        51 57   15   09    EF  1100  3963          EXTZV   #VA$V_VPN,#VA$S_VPN,R7,R1 ; Get virtual page number
         50   00000000'GF    D0  1105  3964          MOVL    G^MMG$GL_SPTBASE,R0     ; Get base address of SPTs
                50   6041    D0  110C  3965          MOVL    (R0)[R1],R0             ; Get the PTE contents
    51  57   FFFFFE00 8F    CB  1110  3966          BICL3   #^C<VA$M_BYTE>,R7,R1    ; Get buffer offset (BA00-BA08)
                              1118  3967          ASSUME  PTE$S_PFN GE 13
        51  0D   09   50    F0  1118  3968          INSV    R0,#9,#13,R1            ; Copy BA09-BA21
                              111D  3969
                              111D  3970 80$:     ;
                              111D  3971          ; Now calculate the physical/virtual address of each of the ring buffer
                              111D  3972          ; entries, saving the addresses in the PHYSICAL ADDRESS/VIRTUAL
                              111D  3973          ; ADDRESS VECTOR
```

```
                     111D  3974
                           ;
50   0166 C6   9E  111D  3975          MOVAB   CDB_G_RRING(R6),R0      ; Get first ring entry
          57   D4  1122  3976          CLRL    R7                      ; Use R7 as ring index
44 A647   51   D0  1124  3977 90$:     MOVL    R1,CDB_L_RRINGPA(R6)[R7]; Save RING physical address
     51   0C   C0  1129  3978          ADDL    #RCV_C_LENGTH,R1        ; Skip to next
7C A647   50   D0  112C  3979          MOVL    R0,CDB_L_RRINGVA(R6)[R7]; Save RING entry virtual address
     50   0C   C0  1131  3980          ADDL    #RCV_C_LENGTH,R0        ; Skip to next
   EC 57   08   F2  1134  3981          AOBLSS  #MAX_C_RCV,R7,90$       ; Loop if more
                     1138  3982
                     1138  3983          ASSUME  CDB_G_XRING_EQ CDB_G_RRING+<<MAX_C_RCV+1>*RCV_C_LENGTH>
44 A647   51   D0  1138  3984          MOVL    R1,CDB_L_RRINGPA(R6)[R7]; Save RING physical address
     51   0C   C0  113D  3985          ADDL    #RCV_C_LENGTH,R1        ; Skip over chained entry
     50   0C   C0  1140  3986          ADDL    #RCV_C_LENGTH,R0        ; Skip over chained entry
                     1143  3987
          57   D4  1143  3988          CLRL    R7
68 A647   51   D0  1145  3989 100$:    MOVL    R1,CDB_L_XRINGPA(R6)[R7]; Save RING physical address
     51   0C   C0  114A  3990          ADDL    #XMT_C_LENGTH,R1        ; Skip to next
009C C647 50   D0  114D  3991          MOVL    R0,CDB_L_XRINGVA(R6)[R7]; Save RING entry virtual address
     50   0C   C0  1153  3992          ADDL    #XMT_C_LENGTH,R0        ; Skip to next
   EB 57   04   F2  1156  3993          AOBLSS  #MAX_C_XMT,R7,100$      ; Loop if more
68 A647   51   D0  115A  3994          MOVL    R1,CDB_L_XRINGPA(R6)[R7]; Save last RING physical address
                     115F  3995          ;
                     115F  3996          ; Initialize receive ring buffer entries
                     115F  3997          ;
     57   08   9A  115F  3998          MOVZBL  #MAX_C_RCV,R7           ; Set number of entries in ring
51   0166 C6   9E  1162  3999          MOVAB   CDB_G_RRING(R6),R1      ; Get address of RING buffer
     8000 8F   B0  1167  4000 110$:     MOVW    #RCV_FLG_M_LAST,-       ; Init flags
          61       116B  4001                  RCV_B_FLAG(R1)          ;
       08 A1   B4  116C  4002          CLRW    RCV_W_STS(R1)           ; Zero status
     51   0C   C0  116F  4003          ADDL    #RCV_C_LENGTH,R1        ; Skip to next entry
       F2 57   F5  1172  4004          SOBGTR  R7,110$                 ; Loop if more
                     1175  4005          ;
                     1175  4006          ; The last entry "chains" back to the first
                     1175  4007          ;
          8000 8F   B0  1175  4008          MOVW    #RCV_FLG_M_LAST,-       ; Init flags
               61       1179  4009                  RCV_B_FLAG(R1)          ;
            44 A6   B0  117A  4010          MOVW    CDB_L_RRINGPA(R6),-     ; Set the chain address
            04 A1       117D  4011                  RCV_W_ADDR(R1)          ;
            46 A6   9B  117F  4012          MOVZBW  CDB_L_RRINGPA+2(R6),-   ; Set high part of chain address
            02 A1       1182  4013                  RCV_W_ADDRHI(R1)        ;
                    A8  1184  4014          BISW    #RCV_DSC_M_CHAIN!-      ; Indicate chain operation
                        1185  4015                   RCV_DSC_M_VALID,-      ;   and valid address
02 A1       C000 8F     1185  4016                   RCV_W_ADDRHI(R1)       ;
                        118A  4017          ;
                        118A  4018          ; Initialize transmit ring buffer entries
                        118A  4019          ;
          57   04   9A  118A  4020          MOVZBL  #MAX_C_XMT,R7           ; Set number of XMIT entries
     51   01D2 C6   9E  118D  4021          MOVAB   CDB_G_XRING(R6),R1      ; Get address of RING buffer
          8000 8F   B0  1192  4022 120$:     MOVW    #XMT_FLG_M_LAST,-       ; Init flags
               61       1196  4023                  XMT_B_FLAG(R1)          ;
            08 A1   B4  1197  4024          CLRW    XMT_W_STS(R1)           ; Zero status
          51   0C   C0  119A  4025          ADDL    #XMT_C_LENGTH,R1        ; Skip to next entry
            F2 57   F5  119D  4026          SOBGTR  R7,120$                 ; Loop if more
                        11A0  4027          ;
                        11A0  4028          ; The last entry "chains" back to the first
                        11A0  4029          ;
          8000 8F   B0  11A0  4030          MOVW    #XMT_FLG_M_LAST,-       ; Init flags
```

```
                61      11A4  4031                    XMT_W_FLAG(R1)          ;
            68 A6   B0  11A5  4032            MOVW    CDB_L_XRINGPA(R6),-     ; Set the chain address
            04 A1       11A8  4033                    XMT_W_ADDR(R1)          ;
            6A A6   9B  11AA  4034            MOVZBW  CDB_L_XRINGPA+2(R6),-   ; Set high part of chain address
            02 A1       11AD  4035                    XMT_W_ADDRHI(R1)        ;
                    A8  11AF  4036            BISW    #XMT_DSC_M_CHAIN!-      ; Indicate chain operation
                        11B0  4037                     XMT_DSC_M_VALID,-      ;   and valid address
      02 A1     C000 8F 11B0  4038                     XMT_W_ADDRHI(R1)       ;
                        11B5  4039 ;
                        11B5  4040 ; Initialize contiguous buffer area for u-VAX I.
                        11B5  4041 ;
                        11B5  4042            CPUDISP <<790,140$>,-
                        11B5  4043                    <780,140$>,-
                        11B5  4044                    <750,140$>,-
                        11B5  4045                    <730,140$>,-
                        11B5  4046                    <UV1,122$>>             ; Initialize buffer area for UV1
                        11CF  4047
                        11CF  4048 122$:                                      ; Init u-VAX I buffer area
        52  0210 C6     D0  11CF  4049            MOVL    CDB_L_UV1BUF(R6),R2 ; Get buffer area address
                09      12  11D4  4050            BNEQ    123$                ; Br if present
                        11D6  4051            POPQ    R6                      ; Restore R6, R7
        50  0124 8F     3C  11D9  4052            MOVZWL  #SS$_INSFMEM,R0     ; Else, return error
                        05  11DE  4053            RSB
                        11DF  4054
                        11DF  4055 123$:  ;
                        11DF  4056        ; Compute physical/virtual address of buffers in buffer area
          52  0C A2      9E  11DF  4057            MOVAB   12(R2),R2           ; Get start of buffer area
    50    52  15   09     EF  11E3  4058            EXTZV   #VA$V_VPN,#VA$S_VPN,R2,R0 ; Get virtual page number
     51    00000000'GF    D0  11E8  4059            MOVL    G^MMG$GL_SPTBASE,R1 ; Get base address of SPTs
        50   6140        D0  11EF  4060            MOVL    (R1)[R0],R0         ; Get the PTE contents
51  52    FFFFFE00 8F     CB  11F3  4061            BICL3   #^C<VA$M_BYTE>,R2,R1 ; Get buffer offset (BA00-BA08)
                        11FB  4062            ASSUME  PTE$S_PFN GE 13
    51  0D  09   50     F0  11FB  4063            INSV    R0,#9,#13,R1        ; Copy BA09-BA21
                50      D4  1200  4064            CLRL    R0                  ; Use R0 as receive buffer index
      00AC C640     51  D0  1202  4065 124$:      MOVL    R1,CDB_L_RCV_PA(R6)[R0] ; Save receive physical address
      00C4 C640     52  D0  1208  4066            MOVL    R2,CDB_L_RCV_VA(R6)[R0] ; Save receive virtual adress
     51   000005EA 8F  C0  120E  4067            ADDL    #MAX_BUFSIZ_UV1,R1  ; Skip to next buffer
     52   000005EA 8F  C0  1215  4068            ADDL    #MAX_BUFSIZ_UV1,R2  ;   ...
        E2 50       05  F2  121C  4069            AOBLSS  #MAX_C_RCVUV1,R0,124$ ; Loop if more
                50      D4  1220  4070            CLRL    R0                  ; Use R0 as transmit buffer index
      00C0 C640     51  D0  1222  4071 125$:      MOVL    R1,CDB_L_XMT_PA(R6)[R0] ; Save transmit physical address
      00D8 C640     52  D0  1228  4072            MOVL    R2,CDB_L_XMT_VA(R6)[R0] ; Save transmit virtual address
     51   000005EA 8F  C0  122E  4073            ADDL    #MAX_BUFSIZ_UV1,R1  ; Skip to next buffer
     52   000005EA 8F  C0  1235  4074            ADDL    #MAX_BUFSIZ_UV1,R2  ;   ...
        E2 50       01  F2  123C  4075            AOBLSS  #MAX_C_XMTUV1,R0,125$ ; Loop if more
                        1240  4076
            54  56      D0  1240  4077 140$:      MOVL    R6,R4               ; Set R4 to CDB address
                        1243  4078            POPQ    R6                  ; Restore R6, R7
                        1246  4079 ;
                        1246  4080 ; Setup fork process to start CDB timer
                        1246  4081 ;
                38      BB  1246  4082            PUSHR   #^M<R3,R4,R5>       ; Save registers
                03      E2  1248  4083            BBSS    #CDB_STS_V_TIMER,-  ; Br if timer already going
          24 024A C4        124A  4084                    CDB_B_STS(R4),150$  ;
                04      88  124E  4085            BISB    #DPT$M_NOUNLOAD,-   ; Do not allow driver to be unloaded
         0000000D'EF        1250  4086                    DPT$TAB+DPT$B_FLAGS ;  while the TQE is active
        55  0218 C4      DE  1255  4087            MOVAL   CDB_L_TQE(R4),R5    ; Get the TQE address
```

```
20 A5   00000000 01312D00 8F   7D  125A  4088          MOVQ    #TQE_DELTA,TQE$Q_DELTA(R5) ; Set the delta time
                        53   1EBD'CF   9E  1266  4089          MOVAB   W^TQE_TIMER,R3             ; Set address of timer wakeup routine
                      2C A5     05     90  126B  4090          MOVB    #TQE$C_SSREPT,TQE$L_RQPID(R5) ; Set the TQE request type
                            00BA     30  126F  4091          BSBW    FORK_TIMER                 ; Create fork process for TQE
                                     1272  4092 ;
                                     1272  4093 ; Get hardware CSR address
                                     1272  4094 ;
                              38     BA  1272  4095 150$:     POPR    #^M<R3,R4,R5>              ; Restore registers
                        52    24 A5   D0  1274  4096          MOVL    UCB$L_CRB(R5),R2           ; Get CRB adddress
                        52    2C B2   D0  1278  4097          MOVL    @CRB$L_INTD+VEC$L_IDB(R2),R2 ; Get CSR address
                                     127C  4098 ;
                                     127C  4099 ; Master reset device
                                     127C  4100 ;
                                     127C  4101          DSBINT  UCB$B_DIPL(R5)             ; Sync access to UCB
                     0E A2    02     B0  1283  4102          MOVW    #XQ_CSR_M_RESET,CSR(R2) ; Master Reset device
                                     1287  4103 ;
                                     1287  4104 ; The master reset will take some time to complete ...
                                     1287  4105 ; so we will delay to give the QNA some time.
                                     1287  4106 ;
                                     1287  4107          TIMEWAIT #1,#XQ_CSR_M_ERR,CSR(R2),W ; Wait for 10 usec, bit should not set
                  0C A2    0050 8F   B0  12AD  4108          MOVW    #^O120,VECTOR(R2)          ; Set vector address
                                     12B3  4109 ;
                                     12B3  4110 ; Copy the Ethernet Hardware Address
                                     12B3  4111 ;
                     0254 C4    62     90  12B3  4112          MOVB    PHYADD0(R2),CDB_G_HWA(R4) ; Save Hardware address
                  0255 C4     02 A2     90  12B8  4113          MOVB    PHYADD1(R2),CDB_G_HWA+1(R4) ;   ...
                  0256 C4     04 A2     90  12BE  4114          MOVB    PHYADD2(R2),CDB_G_HWA+2(R4) ;   ...
                  0257 C4     06 A2     90  12C4  4115          MOVB    PHYADD3(R2),CDB_G_HWA+3(R4) ;   ...
                  0258 C4     08 A2     90  12CA  4116          MOVB    PHYADD4(R2),CDB_G_HWA+4(R4) ;   ...
                  0259 C4     0A A2     90  12D0  4117          MOVB    PHYADD5(R2),CDB_G_HWA+5(R4) ;   ...
                                     12D6  4118 ;
                                     12D6  4119 ; Set CSR mode and enable receiver
                                     12D6  4120 ;
                     0E A2    02     AA  12D6  4121          BICW    #XQ_CSR_M_RESET,CSR(R2) ; Clear the master reset
                                     12DA  4122
                                     12DA  4123          ASSUME  NMA$C_LINCN_NOR EQ 0
                                     12DA  4124          ASSUME  NMA$C_LINCN_LOO EQ 1
                  06 024D C4         E8  12DA  4125          BLBS    CDB_B_CON(R4),160$         ; Br if LOOPBACK is enabled
0E A2               0100 8F         A8  12DF  4126          BISW    #XQ_CSR_M_ILOOP,CSR(R2) ; Else, DISABLE LOOPBACK
0E A2               0040 8F         A8  12E5  4127 160$:    BISW    #XQ_CSR_M_INTENA,CSR(R2); Enable transmit interrupts
                        44 A4       B0  12EB  4128          MOVW    CDB_L_RRINGPA(R4),-        ; Set address of receive list entry
                        04 A2           12EE  4129          RCVLIST(R2)                        ;
                        46 A4       90  12F0  4130          MOVB    CDB_L_RRINGPA+2(R4),-      ;  and high order part
                        06 A2           12F3  4131          RCVLIST1(R2)
                        68 A4       B0  12F5  4132          MOVW    CDB_L_XRINGPA(R4),-        ; Set address of transmit list entry
                        08 A2           12F8  4133          XMTLIST(R2)                        ;
                        6A A4       90  12FA  4134          MOVB    CDB_L_XRINGPA+2(R4),-      ;  and high order part
                        0A A2           12FD  4135          XMTLIST1(R2)
                                     12FF  4136          ENBINT                             ; Re-enable interrupts
                                     1302  4137 ;
                                     1302  4138 ; Initialize QNA mode.
                                     1302  4139 ;
                         006A        30  1302  4140          BSBW    SETUP_MODE                 ; Setup the QNA mode
                      23 50          E9  1305  4141          BLBC    R0,180$                    ; Exit if error
                         00          90  1308  4142          MOVB    #XQ_FC_V_INIT,-            ; Set function request
                      20 A2              130A  4143                  CXB$B_XQ_FUNC(R2)          ;
                         9E        16  130C  4144          JSB     @(SP)+                     ; Complete function request
```

```
                     130E  4145          SETBIT  #CDB_STS_V_RUN,-          ; Indicate QNA is running
                     130E  4146                  CDB_B_STS(R4)            ;
                     1314  4147          SETBIT  #UCB$V_XQ_RUN,-          ; Indicate UNIT is running
                     1314  4148                  UCB$W_DEVSTS(R5)         ;
50   24 A5     D0    1319  4149          MOVL    UCB$L_CRB(R5),R0         ; Get CRB adddress
50   2C B0     D0    131D  4150          MOVL    @CRB$L_INTD+VEC$L_IDB(R0),R0 ; Get CSR address
0E A0   01     A8    1321  4151          BISW    #XQ_CSR_M_RCVENA,CSR(R0) ; Enable receive interrupts
     0116      30    1325  4152          BSBW    FILCRCVLIST              ; Start the receives
   50   01     9A    1328  4153          MOVZBL  S^#SS$_NORMAL,R0         ; Return success
                05    132B  4154 180$:    RSB                              ; Return to caller
                     132C  4155
                     132C  4156 ;++
                     132C  4157 ; FORK_TIMER - Routine to create a fork process to start a timer
                     132C  4158 ;
                     132C  4159 ; This routine starts up a FORK process which is used to start a timer.
                     132C  4160 ;
                     132C  4161 ; Inputs:
                     132C  4162 ;
                     132C  4163 ;       R3 = Address of system routine to handle the timer expiration.
                     132C  4164 ;       R5 = Address of TQE block
                     132C  4165 ;
                     132C  4166 ;       IPL = Greater than Queueast IPL
                     132C  4167 ;
                     132C  4168 ; Outputs:
                     132C  4169 ;
                     132C  4170 ;       Fork process is started.
                     132C  4171 ;       R3, R4 are destroyed by EXE$FORK
                     132C  4172 ;
                     132C  4173 ;--
                     132C  4174 FORK_TIMER:                               ; Create fork process to start timer
0A A5     0F    90   132C  4175          MOVB    #DYN$C_TQE,FKB$B_TYPE(R5) ; Set structure type
0B A5     06    90   1330  4176          MOVB    #IPL$_QUEUEAST,FKB$B_FIPL(R5) ; Set IPL of fork process
       3D'AF    9F   1334  4177          PUSHAB  B^START_TIMER            ; Push address of fork process
00000000'GF    17   1337  4178          JMP     G^EXE$FORK               ; Create fork process to start timer
                     133D  4179
                     133D  4180 ;++
                     133D  4181 ; START_TIMER - Fork process to start the CDB timer
                     133D  4182 ;
                     133D  4183 ; This routine starts up the CDB which is used to monitor the QNA controller
                     133D  4184 ; for proper operation.
                     133D  4185 ;
                     133D  4186 ; Inputs:
                     133D  4187 ;
                     133D  4188 ;       R5 = Address of TQE block
                     133D  4189 ;
                     133D  4190 ;       IPL = Queueast IPL
                     133D  4191 ;
                     133D  4192 ; Implicit inputs:
                     133D  4193 ;
                     133D  4194 ;       TQE$Q_DELTA(R5) = Delta time interval
                     133D  4195 ;       TQE$L_RQPID(R5) = TQE request type (SSSNGL or SSREPT)
                     133D  4196 ;
                     133D  4197 ; Outputs:
                     133D  4198 ;
                     133D  4199 ;       R0-R3 are destroyed.
                     133D  4200 ;       TQE element added to timer queue
                     133D  4201 ;
```

```
                         133D  4202 ;--
                         133D  4203 START_TIMER:
                         133D  4204          DSBINT  #IPL$_TIMER             ; Raise IPL
    0C A5    10 A5   D0  1343  4205          MOVL    FKB$L_FR3(R5),TQE$L_FPC(R5) ; Set address of timer wakeup
             2C A5   90  1348  4206          MOVB    TQE$L_RQPID(R5),-       ; Set TQE request type
             0B A5           134B  4207                  TQE$B_RQTYPE(R5)    ;
       50    20 A5   7D  134D  4208          MOVQ    TQE$Q_DELTA(R5),R0      ; Get delta time
50  00000000'GF   C0  1351  4209          ADDL    G^EXE$GQ_SYSTIME,R0     ; Add in current time
51  00000004'GF   D8  1358  4210          ADWC    G^EXE$GQ_SYSTIME+4,R1   ;       ...
    00000000'GF   16  135F  4211          JSB     G^EXE$INSTIMQ           ; Insert element on timer queue
                  1365  4212          ENBINT                          ; Restore IPL
                  05  1368  4213          RSB
```

```
                        1369  4215          .SBTTL  SETUP_MODE - SETUP THE TRANSMIT BUFFER TO INIT QNA
                        1369  4216 ;++
                        1369  4217 ; SETUP_MODE - SETUP THE TRANSMIT BUFFER TO INIT QNA
                        1369  4218 ;
                        1369  4219 ; Functional description:
                        1369  4220 ;
                        1369  4221 ; This routine initializes the TRANSMIT buffer the sets up the QNA operating
                        1369  4222 ; mode.
                        1369  4223 ;
                        1369  4224 ; Inputs:
                        1369  4225 ;
                        1369  4226 ;       R3 = IRP address
                        1369  4227 ;       R4 = CDB address
                        1369  4228 ;       R5 = UCB address
                        1369  4229 ;
                        1369  4230 ;       IPL = FIPL
                        1369  4231 ;
                        1369  4232 ; Outputs:
                        1369  4233 ;
                        1369  4234 ;       R0 = status of request
                        1369  4235 ;
                        1369  4236 ;       R1,R2 are destroyed
                        1369  4237 ;--
                        1369  4238 SETUP_ERR:                                ; Setup error
     50  0124 8F    3C  1369  4239          MOVZWL  #SS$_INSFMEM,R0         ; Set error return
                    05  136E  4240          RSB                             ; Return to caller
                        136F  4241
                        136F  4242 SETUP_MODE:
                    53  DD  136F  4243          PUSHL   R3                      ; Save R3
              51    00C8 8F  3C  1371  4244          MOVZWL  #CXB$C_HEADER+INIT_C_BUFSIZE,R1 ; Size of init buffer
               00000000'GF  16  1376  4245          JSB     G^EXE$ALONONPAGED       ; Allocate the SETUP Transmit buffer
                    53 8ED0  137C  4246          POPL    R3                      ; Restore R3
                 E7 50  E9  137F  4247          BLBC    R0,SETUP_ERR            ; Exit if error
                    3C  BB  1382  4248          PUSHR   #^M<R2,R3,R4,R5>        ; Save registers
62   51   FF 8F   62  00  2C  1384  4249          MOVC5   #0,(R2),#-1,R1,(R2)     ; Fill structure with BROADCAST!
                    3C  BA  138B  4250          POPR    #^M<R2,R3,R4,R5>        ; Restore registers
                        138D  4251          ;
                        138D  4252          ; Initialize buffer to look like a CXB
                        138D  4253          ;
              0094 C3   52  D0  138D  4254          MOVL    R2,IRP$L_XQ_SETUP(R3)   ; Save address of setup buffer
                        1392  4255          ASSUME  CXB$B_TYPE EQ CXB$W_SIZE+2
                        1392  4256          ASSUME  CXB$B_CODE EQ CXB$B_TYPE+1
                    D0  1392  4257          MOVL    #<DYN$C_CXB@16>!-       ; Set size and type of structure
                        1393  4258                  CXB$C_HEADER+INIT_C_BUFSIZE,-
     08 A2   001B00C8 8F        1393  4259                  CXB$W_SIZE(R2)
        1A A2    0080 8F   B0  139A  4260          MOVW    #INIT_C_BUFSIZE,CXB$W_BCNT(R2) ; Set size of transfer
                        13A0  4261          ;
                        13A0  4262          ; Initialize QNA mode word
                        13A0  4263          ;
                        13A0  4264          ASSUME  NMA$C_STATE_ON EQ 0
                        13A0  4265          ASSUME  NMA$C_STATE_OFF EQ 1
              0248 C4   B4  13A0  4266          CLRW    CDB_W_MODE(R4)          ; Init mode word
           05 024B C4   E8  13A4  4267          BLBS    CDB_B_PRM(R4),10$       ; Br if promiscuous state is OFF
                    02  A8  13A9  4268          BISW    #CDB_MOD_M_PROM,-       ; Else, enable promiscuous mode
              0248 C4        13AB  4269                  CDB_W_MODE(R4)          ;
           05 024C C4   E8  13AE  4270 10$:     BLBS    CDB_B_MLT(R4),20$       ; Br if multicast state is OFF
                    01  A8  13B3  4271          BISW    #CDB_MOD_M_MULTI,-      ; Else, enable all MULTICASTS
```

```
                0248 C4          13B5  4272                 CDB_W_MODE(R4)                  ;
      1A A2     0248 C4    A8    13B8  4273 20$:    BISW    CDB_W_MODE(R4),CXB$W_BCNT(R2) ; Set mode bits
           18 A2      3A    B0    13BE  4274          MOVW    #CXB$T_T_DATA,CXB$W_BOFF(R2) ; Set offset to start of data
           24 A2      53    D0    13C2  4275          MOVL    R3,CXB$L_T_IRP(R2)      ; Save IRP address in CXB
                                  13C6  4276          ;
                                  13C6  4277          ; Initialize physical address
                                  13C6  4278          ;
           50    024E C4    9E    13C6  4279          MOVAB   CDB_G_PHA(R4),R0        ; Point to Physical Address
             51     3B A2    9E    13CB  4280          MOVAB   CXB$T_T_DATA+1(R2),R1   ; Point to setup buffer (skip 1st col)
                 61     80    90    13CF  4281          MOVB    (R0)+,(R1)              ; Stuff the physical address
              08 A1     80    90    13D2  4282          MOVB    (R0)+,8(R1)             ;       ...
              10 A1     80    90    13D6  4283          MOVB    (R0)+,16(R1)            ;       ...
              18 A1     80    90    13DA  4284          MOVB    (R0)+,24(R1)            ;       ...
              20 A1     80    90    13DE  4285          MOVB    (R0)+,32(R1)            ;       ...
              28 A1     80    90    13E2  4286          MOVB    (R0)+,40(R1)            ;       ...
                                  13E6  4287          ;
                                  13E6  4288          ; Initialize multicast addresses
                                  13E6  4289          ;
                                  13E6  4290          PUSHQ   R2                      ; Save setup buffer, IRP address
           52    0260 C4    9A    13E9  4291          MOVZBL  CDB_B_MULTI(R4),R2      ; Get number of multicast addresses
                       38    13    13EE  4292          BEQL    80$                     ; Br if none
                 52    0C    91    13F0  4293          CMPB    #12,R2                  ; Is count okay?
                       04    1E    13F3  4294          BGEQU   30$                     ; Br if yes
                                  13F5  4295          BUG_CHECK NOBUFPCKT,FATAL       ; Else, error
                                  13F9  4296
                 53    06    9A    13F9  4297 30$:    MOVZBL  #6,R3                   ; Only 6 slots left in first half of
                                  13FC  4298                                          ;  setup buffer
           50    0262 C4    9E    13FC  4299          MOVAB   CDB_G_MULTI(R4),R0      ; Point to multicast address list
                       09    11    1401  4300          BRB     50$                     ; Start with first half
                                  1403  4301          ;
                                  1403  4302          ; Check if first half of setup buffer is full
                                  1403  4303          ;
                    06 53    F5    1403  4304 40$:    SOBGTR  R3,50$                  ; Br if first half of buffer still open
                 51    39    C0    1406  4305          ADDL    #64-7,R1                ; Skip to second half of buffer
                 53    06    9A    1409  4306          MOVZBL  #6,R3                   ; Reset count for second half
                                  140C  4307                                          ; Leave last address as BROADCAST
                                  140C  4308          ;
                                  140C  4309          ; Store multicast addresses
                                  140C  4310          ;
                       51    D6    140C  4311 50$:    INCL    R1                      ; Skip to next column
                 61    80    90    140E  4312          MOVB    (R0)+,(R1)              ; Store multicast address
              08 A1     80    90    1411  4313          MOVB    (R0)+,8(R1)             ;       ...
              10 A1     80    90    1415  4314          MOVB    (R0)+,16(R1)            ;       ...
              18 A1     80    90    1419  4315          MOVB    (R0)+,24(R1)            ;       ...
              20 A1     80    90    141D  4316          MOVB    (R0)+,32(R1)            ;       ...
              28 A1     80    90    1421  4317          MOVB    (R0)+,40(R1)            ;       ...
                 DB 52    F5    1425  4318          SOBGTR  R2,40$                  ; Br if more
                                  1428  4319
                                  1428  4320 80$:    POPQ    R2                      ; Restore setup buffer, IRP address
                 50    01    9A    142B  4321          MOVZBL  #SS$_NORMAL,R0          ; Return success
                       9E    16    142E  4322          JSB     @(SP)+                  ; Call back caller as co-routine
            00E0 D4    62    0E    1430  4323          INSQUE  (R2),@CDB_Q_XMTREQ+4(R4); Insert request on xmit queue
                                  1435  4324          SETBIT  #CDB_STS_V_SETUP,-      ; Indicate that SETUP is in progress
                                  1435  4325                  CDB_B_STS(R4)           ;
                    F055    31    143B  4326          BRW     XMT_ACT_START           ; Startup the XMIT process
                                  143E  4327
```

```
                     143E  4329          .SBTTL  FILLRCVLIST - FILL RECEIVE BUFFER LIST
                     143E  4330 ;++
                     143E  4331 ; FILLRCVLIST - FILL RECEIVE BUFFER LIST
                     143E  4332 ;
                     143E  4333 ; Functional description:
                     143E  4334 ;
                     143E  4335 ; This routine fills the receive buffer list up to the quota allocated
                     143E  4336 ; at unit initialization.  It also gives any receive buffers allocated
                     143E  4337 ; to the receiver.
                     143E  4338 ;
                     143E  4339 ; Inputs:
                     143E  4340 ;
                     143E  4341 ;       R2 = Buffer Address (ADDRCVLIST ONLY)
                     143E  4342 ;       R4 = CDB address
                     143E  4343 ;
                     143E  4344 ;       IPL = FIPL
                     143E  4345 ;
                     143E  4346 ; Outputs:
                     143E  4347 ;
                     143E  4348 ;       R0-R2 is destroyed.
                     143E  4349 ;       All other registers are preserved.
                     143E  4350 ;
                     143E  4351 ;--
                     143E  4352          .ENABL  LSB
                     143E  4353 FILLRCVLIST::                            ; Fill receive buffer list
52   D4              143E  4354          CLRL    R2                      ; No buffer here
                     1440  4355
                     1440  4356 ADDRCVLIST::                            ; Add a buffer to the receive list
          38   BB    1440  4357          PUSHR   #^M<R3,R4,R5>           ; Save registers
     55   0118 C4   D0    1442  4358          MOVL    CDB_L_UCB0(R4),R5       ; Get UCB address of unit #0
          01   E1    1447  4359          BBC     #CDB_STS_V_RUN,-        ; If BC device not running
                     1449  4360                  CDB_B_STS(R4),40$
     5D   024A C4              144D  4361 10$:    CMPW    CDB_W_BSZ(R4),-         ; Can new block be allocated ?
          0110 C4   B1    1451  4362                  CDB_W_QUOTA(R4)
          0112 C4             1454  4363          BGTRU   35$                     ; If GTRU then no, stop loop
               3B   1A    1456  4364          CLRL    R1                      ; Zero size
               51   D4    1458  4365          ADDW3   #CXB$C_HEADER+-         ; Determine block size needed
                     1459  4366                  CXB$C_TRAILER,-         ;
51   0110 C4   004C 8F        1459  4367                  CDB_W_BSZ(R4),R1        ;
               52   D5    1460  4368          TSTL    R2                      ; Buffer already allocated?
               09   12    1462  4369          BNEQ    20$                     ; Br if so
     00000000'GF   16    1464  4370          JSB     G^EXE$ALONONPAGED       ; Allocate the memory
          1C   50    E9    146A  4371          BLBC    R0,30$                  ; If failure then done
          0110 C4   A2    146D  4372 20$:     SUBW    CDB_W_BSZ(R4),-         ; Subtract from quota
          0112 C4             1471  4373                  CDB_W_QUOTA(R4)         ;
     08   A2   51   B0    1474  4374          MOVW    R1,CXB$W_SIZE(R2)       ; Insert size
               1B   90    1478  4375 25$:     MOVB    S^#DYN$C_CXB,-          ; Insert type
          0A   A2             147A  4376                  CXB$B_TYPE(R2)          ;
     20   A2   02   90    147C  4377          MOVB    #XQ_FC_V_RECV,CXB$B_XQ_FUNC(R2) ; Set function request
     00F4 C4   62   0E    1480  4378          INSQUE  (R2),CDB_Q_RCVBUF(R4)   ; Insert block on list
               52   D4    1485  4379          CLRL    R2                      ; No more buffers given
               C4   11    1487  4380          BRB     10$                     ; Continue
                     1489  4381 ;
                     1489  4382 ; Buffer allocation failure
                     1489  4383 ;
                     1489  4384 30$:     SETBIT  #XM$V_STS_BUFFAIL,-     ; Set buffer alloc failure
                     1489  4385                  CDB_L_DEVDEPEND(R4)     ;
```

```
                24   11  148F  4386          BRB     50$                     ; And give any receives to device
                                 1491  4387
                                 1491  4388 35$:     CLRBIT  #XM$V_STS_BUFFAIL,-     ; Clear buffer alloc failure
                                 1491  4389          CDB_L_DEVDEPEND(R4)     ;
                52   D5  1497  4390          TSTL    R2                      ; Any buffer?
                1A   13  1499  4391          BEQL    50$                     ; Br if not
          02F2  C4   91  149B  4392          CMPB    CDB_B_AQUOTA(R4),-      ; Can we use the additional quota?
          02F3  C4        149F  4393          CDB_B_MQUOTA(R4)        ;
                06   1E  14A2  4394          BGEQU   40$                     ; Br if not
          02F2  C4   96  14A4  4395          INCB    CDB_B_AQUOTA(R4)        ; Else, increment the additional
                CE   11  14A8  4396          BRB     25$                     ; Use buffer, but don't let
                                 14AA  4397                                  ;  QUOTA go negative
                                 14AA  4398
           50   52   D0  14AA  4399 40$:     MOVL    R2,R0                   ; Get address of buffer
                06   13  14AD  4400          BEQL    50$                     ; Br if none
   00000000'GF   16  14AF  4401          JSB     G^COM$DRVDEALMEM        ; Deallocate buffer
                                 14B5  4402
                03   10  14B5  4403 50$:     BSBB    START_RECEIVE           ; Start the receives
                38   BA  14B7  4404          POPR    #^M<R3,R4,R5>           ; Restore registers
                     05  14B9  4405          RSB
                                 14BA  4406          .DSABL  LSB
```

```
                        14BA  4408          .SBTTL  START_RECEIVE - START ANY RECEIVE REQUESTS PENDING
                        14BA  4409 ;++
                        14BA  4410 ; START_RECEIVE - START ANY RECEIVE REQUESTS
                        14BA  4411 ;
                        14BA  4412 ; Function[illegible]n:
                        14BA  4413 ;
                        14BA  4414 ; This rou[illegible]to start any receives that may be pending.  This
                        14BA  4415 ; involves [illegible]ree receive buffer, mapping it, and loading it's
                        14BA  4416 ; address and size i[illegible]e device.
                        14BA  4417 ;
                        14BA  4418 ; Inputs:
                        14BA  4419 ;
                        14BA  4420 ;       R4 = CDB address
                        14BA  4421 ;       R5 = UCB address of unit # 0
                        14BA  4422 ;
                        14BA  4423 ;       IPL = FIPL
                        14BA  4424 ;
                        14BA  4425 ; Outputs:
                        14BA  4426 ;
                        14BA  4427 ;       R0-R3 are destroyed.
                        14BA  4428 ;       All other registers are preserved.
                        14BA  4429 ;
                        14BA  4430 ;--
                        14BA  4431
                        14BA  4432 START_RECEIVE::                              ; Start receive operation
                        14BA  4433          PUSHQ   R6                          ; Save R6, R7
                  01 E1 14BD  4434          BBC     #CDB_STS_V_RUN,-            ; Br if device is not running
              29 024A C4 14BF  4435                  CDB_B_STS(R4),10$           ;
                        14C3  4436          ;
                        14C3  4437          ; For u-VAX I, we will not use map registers.
                        14C3  4438          ;
                        14C3  4439          CPUDISP <<790,5$>,-
                        14C3  4440                   <780,5$>,-
                        14C3  4441                   <750,5$>,-
                        14C3  4442                   <730,5$>,-
                        14C3  4443                   <UV1,40$>>                 ; For u-VAX I, use alternate path
                        14DD  4444
          57 1A A4 07 00 EB 14DD  4445 5$:        FFC     #0,#MAX_C_RCV-1,CDB_B_RCVMAP(R4),R7 ; Get a free mapping slot
                     07 13  14E3  4446          BEQL    10$                         ; Br if none - just exit
                53 00F4 D4  0F  14E5  4447          REMQUE  @CDB_Q_RCVBUF(R4),R3    ; Get a free buffer
                     04 1C  14EA  4448          BVC     20$                         ; Br if buffer found
                        14EC  4449 10$:      POPQ    R6                          ; Restore R6, R7
                     05 14EF  4450          RSB                                 ; Return to caller
                        14F0  4451 ;
                        14F0  4452 ; Mark slot in use and create buffer address/character count image
                        14F0  4453 ; in receive buffer and load UNIBUS adapter map registers.
                        14F0  4454 ;
                        14F0  4455 20$:      SETBIT  R7,CDB_B_RCVMAP(R4)         ; Mark slot in use
             22 A3 57 90 14F5  4456          MOVB    R7,CXB$B_XQ_SLOT(R3)        ; Save mapping slot index
             52 24 A5 D0 14F9  4457          MOVL    UCB$L_CRB(R5),R2            ; Get CRB address
                        14FD  4458 ;
                        14FD  4459 ; Find next ring entry and insert data
                        14FD  4460 ;
             56 19 A4 9A 14FD  4461          MOVZBL  CDB_B_NEXTRCV(R4),R6        ; Get next ring entry
                19 A4 96 1501  4462          INCB    CDB_B_NEXTRCV(R4)           ; Bump ring pointer
                F8 8F 8A 1504  4463          BICB    #^C<MAX_C_RCV-1>,-          ; Modulo rcv ring size
                19 A4       1507  4464                  CDB_B_NEXTRCV(R4)       ;
```

```
                                                K 13
                23 A3    56   90  1509  4465          MOVB    R6,CXB$B_XQ_RING(R3)    ; Save ring entry number
             56    7C A446   D0  150D  4466          MOVL    CDB_L_RRINGVA(R4)[R6],R6; Get virtual address of ring entry
                       66   B4  1512  4467          CLRW    RCV_W_FLAG(R6)          ; Zero the FLAG word
                  8000 8F   B0  1514  4468          MOVW    #RCV_STS_M_LAST,-       ; Init the STATUS word
                    08 A6       1518  4469                  RCV_W_STS(R6)           ;
                                151A  4470          PUSHQ   R3                      ; Save IRP, CDB address
                                151D  4471 ;
                                151D  4472 ; The QNA's receive buffer size must be a multiple of 2
                                151D  4473 ;
                                151D  4474          ASSUME  <XQ_C_HEADER+XQ_C_CNTSIZ & 1 > EQ 0
                       10   A1  151D  4475          ADDW3   #XQ_C_HEADER+XQ_C_CNTSIZ,- ; Calculate message length
                50    0110 C4   151F  4476                  CDB_W_BSZ(R4),R0        ;                  ...
             50    50    FF 8F   78  1523  4477          ASHL    #-1,R0,R0               ; Convert byte count to WORD count
                   06 A6    50   AE  1528  4478          MNEGW   R0,RCV_W_LEN(R6)        ; Store length (2's complement)
                   51    38 A3   9E  152C  4479          MOVAB   CXB$T_R_DATA(R3),R1     ; Get receive buffer data addr
                   04 A6    51   B0  1530  4480          MOVW    R1,RCV_W_ADDR(R6)       ; Set BA0-BA8
                57    1C A447   DE  1534  4481          MOVAL   CDB_L_RCVMAP(R4)[R7],R7 ; Get mapping info slot address
          04 A6    07    09    67   F0  1539  4482          INSV    (R7),#9,#7,RCV_W_ADDR(R6) ; Set BA9-BA15 from map reg
             50    67    06    07   EF  153F  4483          EXTZV   #7,#6,(R7),R0           ; Get BA16-BA21 also
                   02 A6    50   B0  1544  4484          MOVW    R0,RCV_W_ADDRHI(R6)     ; Set BA16-BA21 & zero descriptor bits
                   0A A6    01   B0  1548  4485          MOVW    #1,RCV_W_LENB(R6)       ; Set low byte not equal to high byte
                            54   D4  154C  4486          CLRL    R4                      ; Use direct data path for rcvs
                                154E  4487          ASSUME  VEC$B_NUMREG EQ VEC$W_MAPREG+2
                   52    02 A7   9A  154E  4488          MOVZBL  2(R7),R2                ; Set number of map registers
                      53    67   3C  1552  4489          MOVZWL  (R7),R3                 ; Set first map register number
                00000000'GF   16  1555  4490          JSB     G^IOC$LOADUBAMAPN       ; Load the map registers
                                155B  4491          POPQ    R3                      ; Restore IRP, CDB address
                                155E  4492 ;
                                155E  4493 ; Disable interrupts and queue request to input queue
                                155E  4494 ;
                                155E  4495          DSBINT  UCB$B_DIPL(R5)          ; Disable interrupts
                      8000 8F   A8  1565  4496          BISW    #RCV_DSC_M_VALID,-      ; Set descriptor bits
                        02 A6       1569  4497                   XMT_W_ADDRHI(R6)       ;  "VALID" buffer address
                      010E C4   96  156B  4498          INCB    CDB_B_RCVCNT(R4)        ; Tally one more receive in progress
                00E8  D4    63   0E  156F  4499          INSQUE  (R3),@CDB_Q_INPUT+4(R4) ; Insert receive buffer on
                                1574  4500                                          ;   input queue
                         03   12  1574  4501          BNEQ    30$                     ; Br if not first entry on queue
                      0085    30  1576  4502          BSBW    LOAD_PORT               ; Request port and give request to QNA
                                1579  4503 30$:      ENBINT                          ; Re-enable interrupts
                      FF5E    31  157C  4504          BRW     5$                      ; Let's try it again
                                157F  4505
                                157F  4506
                                157F  4507 40$:      ;********************
                                157F  4508          ; For u-VAX I, only.
                                157F  4509          ;********************
                                157F  4510
                                157F  4511          ASSUME  MAX_C_RCV LE 8
                                157F  4512          ASSUME  MAX_C_RCVUV1 LT MAX_C_RCV
          57    1A A4    05    00   EB  157F  4513          FFC     #0,#MAX_C_RCVUV1,CDB_B_RCVMAP(R4),R7 ; Get a free mapping slot
                         07   13  1585  4514          BEQL    50$                     ; Br if none - just exit
                   53    00F4 D4   0F  1587  4515          REMQUE  @CDB_Q_RCVBUF(R4),R3    ; Get a free buffer
                         04   1C  158C  4516          BVC     60$                     ; Br if buffer found
                                158E  4517 50$:      POPQ    R6                      ; Restore R6, R7
                              05    1591  4518          RSB                             ; Return to caller
                                1592  4519 ;
                                1592  4520 ; Mark slot in use and create buffer address/character count image
                                1592  4521 ; in receive buffer and load UNIBUS adapter map registers.
```

```
                        1592  4522 ;
                        1592  4523 60$:    SETBIT  R7,CDB_B_RCVMAP(R4)     ; Mark slot in use
      22 A3   57   90   1597  4524         MOVB    R7,CXB$B_XQ_SLOT(R3)    ; Save mapping slot index
                        159B  4525 ;
                        159B  4526 ; Find next ring entry and insert data
                        159B  4527 ;
      56   19 A4   9A   159B  4528         MOVZBL  CDB_B_NEXTRCV(R4),R6    ; Get next ring entry
           19 A4   96   159F  4529         INCB    CDB_B_NEXTRCV(R4)       ; Bump ring pointer
           F8 8F   8A   15A2  4530         BICB    #^C<MAX_C_RCV-1>,-      ; Modulo rcv ring size
           19 A4        15A5  4531                 CDB_B_NEXTRCV(R4)       ;
      23 A3   56   90   15A7  4532         MOVB    R6,CXB$B_XQ_RING(R3)    ; Save ring entry number
   56    7C A446   D0   15AB  4533         MOVL    CDB_L_RRINGVA(R4)[R6],R6; Get virtual address of ring entry
              66   B4   15B0  4534         CLRW    RCV_W_FLAG(R6)          ; Zero the FLAG word
         8000 8F   B0   15B2  4535         MOVW    #RCV_STS_M_LAST,-       ; Init the STATUS word
           08 A6        15B6  4536                 RCV_W_STS(R6)           ;
                        15B8  4537 ;
                        15B8  4538 ; The QNA's receive buffer size must be a multiple of 2
                        15B8  4539 ;
                        15B8  4540         ASSUME  <XQ_C_HEADER+XQ_C_CNTSIZ & 1 > EQ 0
              10   A1   15B8  4541         ADDW3   #XQ_C_HEADER+XQ_C_CNTSIZ,- ; Calculate message length
   50    0110 C4        15BA  4542                 CDB_W_BSZ(R4),R0        ;                ...
50    50    FF 8F   78   15BE  4543         ASHL    #-1,R0,R0               ; Convert byte count to WORD count
      06 A6   50   AE   15C3  4544         MNEGW   R0,RCV_W_LEN(R6)        ; Store length (2's complement)
 50    00AC C447   D0   15C7  4545         MOVL    CDB_L_RCV_PA(R4)[R7],R0 ; Get receive buffer physical address
      04 A6   50   B0   15CD  4546         MOVW    R0,RCV_W_ADDR(R6)       ; Set BA00-BA15
50    50    F0 8F   78   15D1  4547         ASHL    #-16,R0,R0              ; Shift down high byte of address
      02 A6   50   B0   15D6  4548         MOVW    R0,RCV_W_ADDRHI(R6)     ; Set BA16-BA21 & zero descriptor bits
      0A A6   01   B0   15DA  4549         MOVW    #1,RCV_W_LENB(R6)       ; Set low byte not equal to high byte
                        15DE  4550 ;
                        15DE  4551 ; Disable interrupts and queue request to input queue
                        15DE  4552 ;
                        15DE  4553         DSBINT  UCB$B_DIPL(R5)          ; Disable interrupts
         8000 8F   A8   15E5  4554         BISW    #RCV_DSC_M_VALID,-      ; Set descriptor bits
           02 A6        15E9  4555                 XMT_W_ADDRHI(R6)        ;  "VALID" buffer address
      010E C4   96   15EB  4556         INCB    CDB_B_RCVCNT(R4)        ; Tally one more receive in progress
00E8  D4   63   0E   15EF  4557         INSQUE  (R3),@CDB_Q_INPUT+4(R4) ; Insert receive buffer on
                        15F4  4558                                         ;   input queue
              02   12   15F4  4559         BNEQ    90$                     ; Br if not first entry on queue
              06   10   15F6  4560         BSBB    LOAD_PORT               ; Request port and give request to QNA
                        15F8  4561 90$:    ENBINT                          ; Re-enable interrupts
            FF81   31   15FB  4562         BRW     40$                     ; Let's try it again
```

```
                     15FE  4564          .SBTTL  LOAD_PORT - LOAD CSR'S WITH COMMAND REQUEST
                     15FE  4565 ;++
                     15FE  4566 ; LOAD_PORT - LOAD CSR'S WITH COMMAND REQUEST
                     15FE  4567 ;
                     15FE  4568 ; Functional description:
                     15FE  4569 ;
                     15FE  4570 ; This routine loads the CSR's and PCBB with a command to process.
                     15FE  4571 ;
                     15FE  4572 ; Inputs:
                     15FE  4573 ;
                     15FE  4574 ;       R4 = CDB address
                     15FE  4575 ;       R5 = UCB address
                     15FE  4576 ;
                     15FE  4577 ;       IPL = DIPL
                     15FE  4578 ;
                     15FE  4579 ; Outputs:
                     15FE  4580 ;
                     15FE  4581 ;       R4,R5 are preserved.
                     15FE  4582 ;
                     15FE  4583 ;       R0-R3 may be destroyed.
                     15FE  4584 ;
                     15FE  4585 ;--
                     15FE  4586 LOAD_PORT::                                  ; Load port command
   52   24 A5   D0  15FE  4587          MOVL    UCB$L_CRB(R5),R2           ; Get CRB address
                     1602  4588          ASSUME  IDB$L_CSR EQ 0
   52   2C B2   D0  1602  4589          MOVL    @CRB$L_INTD+VEC$L_IDB(R2),R2 ; Get CSR address
53  00E4 D4   0F  1606  4590          REMQUE  @CDB_Q_INPUT(R4),R3        ; Get next CXB
         01   1C  160B  4591          BVC     20$                        ; Br if got one
         05   1D  160D  4592 10$:     RSB                                ; Return to caller
              160E  4593
   1B   0A A3   91  160E  4594 20$:     CMPB    CXB$B_TYPE(R3),#DYN$C_CXB ; Is this a CXB?
         04   13  1612  4595          BEQL    30$                        ; Br if yes
              1614  4596          BUG_CHECK NOBUFPCKT,FATAL          ; Fatal error - what is it???
              1618  4597 ;
              1618  4598 ; Dispatch of CXB request
              1618  4599 ;
              1618  4600 30$:     $DISPATCH       CXB$B_XQ_FUNC(R3),TYPE=B,- ; Dispatch on function request
              1618  4601          <-      ;function       action
              1618  4602                  <XQ_FC_V_XMIT   40$>,-  ; XMIT requested
              1618  4603                  <XQ_FC_V_INIT   40$>,-  ; INIT QNA requested
              1618  4604                  <XQ_FC_V_STOP   40$>,-  ; STOP QNA requested
              1618  4605                  <XQ_FC_V_CHMODE 40$>,-  ; Change mode requested
              1618  4606                  <XQ_FC_V_RECV   60$>,-  ; RECV requested
              1618  4607          >
              162B  4608
              162B  4609          BUG_CHECK NOBUFPCKT,FATAL          ; Fatal error - not a valid IRP
              162F  4610
              162F  4611 ;
              162F  4612 ; XMIT request
              162F  4613 ;
              162F  4614 ;   If the QNA has invalidated the TRANSMIT RING, then we must reset the
              162F  4615 ;   starting address of the ring list to point to the current entry.
              162F  4616 ;
   0E A2   10   B3  162F  4617 40$:     BITW    #XQ_CSR_M_XMTINV,CSR(R2); Is the transmit ring still valid?
         16   13  1633  4618          BEQL    50$                        ; Br if yes, all done
   51   23 A3   9A  1635  4619          MOVZBL  CXB$B_XQ_RING(R3),R1       ; Else, get ring entry number
51   68 A441   D0  1639  4620          MOVL    CDB_L_XRINGPA(R4)[R1],R1   ; Get the buffer mapping value
```

```
                08 A2    51    B0  163E  4621          MOVW    R1,XMTLIST(R2)          ; Set address of transmit list entry
51              51    F0 8F    78  1642  4622          ASHL    #-16,R1,R1              ; Shift down high order address bits
                0A A2    51    90  1647  4623          MOVB    R1,XMTLST1(R2)          ;                ...
             00F0 D4     63    0E  164B  4624 50$:      INSQUE  (R3),@CDB_Q_XMTPND+4(R4) ; Insert CXB on WAITING queue
                                    1650  4625 ;;&&     BITW    #XQ_CSR_M_CAR,CSR(R2)   ; Is carrier present?
                                    1650  4626 ;;&&     BEQL    55$                     ; Br if not, okay - else
                                    1650  4627 ;;&&     INCC    CDB_L_BIDCTR(R4)        ; Count blocks sent - initially deferred
             020E C4     05    90  1650  4628 55$:      MOVB    #XMT_TIM,CDB_B_TIM_XMT(R4) ; Start xmit timer
                         05    05  1655  4629          RSB                             ; Return to caller
                                    1656  4630 ;
                                    1656  4631 ; RECV request
                                    1656  4632 ;
                                    1656  4633 ;   If the QNA has invalidated the RECEIVE RING, then we must reset the
                                    1656  4634 ;   starting address of the ring list to point to the current entry.
                                    1656  4635 ;
                0E A2    20    B3  1656  4636 60$:      BITW    #XQ_CSR_M_RCVINV,CSR(R2); Is the receive ring still valid?
                         16    13  165A  4637          BEQL    70$                     ; Br if yes, all done
                   51    23 A3    9A  165C  4638          MOVZBL  CXB$B_XQ_RING(R3),R1    ; Else, get ring entry number
             51    44 A441       D0  1660  4639          MOVL    CDB_L_RRINGPA(R4)[R1],R1; Get the buffer mapping value
                04 A2    51    B0  1665  4640          MOVW    R1,RCVLIST(R2)          ; Set address of receive list entry
51              51    F0 8F    78  1669  4641          ASHL    #-16,R1,R1              ; Shift down high order address bits
                06 A2    51    90  166E  4642          MOVB    R1,RCVLST1(R2)          ;                ...
             0100 D4     63    0E  1672  4643 70$:      INSQUE  (R3),@CDB_Q_RCVPND+4(R4) ; Insert CXB on WAITING queue
                         05    05  1677  4644          RSB                             ; Return to caller
                                    1678  4645
```

```
                     1678  4647          .SBTTL  QNA_INTR - QNA INTERRUPT SERVICE ROUTINE
                     1678  4648 ;++
                     1678  4649 ; QNA_INTR - QNA INTERRUPT SERVICE ROUTINE
                     1678  4650 ;
                     1678  4651 ; Functional description:
                     1678  4652 ;
                     1678  4653 ; This routine services the interrupts generated by the QNA for completion
                     1678  4654 ; of requests.
                     1678  4655 ;
                     1678  4656 ; Inputs:
                     1678  4657 ;
                     1678  4658 ;       00(SP) = ADDRESS OF UNIT IDB ADDRESS
                     1678  4659 ;
                     1678  4660 ;       R0,R1,R2,R3,R4,R5 ARE AT 04(SP) TO 1C(SP)
                     1678  4661 ;
                     1678  4662 ;
                     1678  4663 ;       IPL = DIPL
                     1678  4664 ;
                     1678  4665 ; Outputs:
                     1678  4666 ;
                     1678  4667 ;       THE INTERRUPT IS DISMISSED
                     1678  4668 ;
                     1678  4669 ; IMPLICIT OUTPUTS:
                     1678  4670 ;
                     1678  4671 ;       A Fork process is started to check ring entries.
                     1678  4672 ;--
                     1678  4673 QNA_INTR::                                ; DEQNA done interrupt
    54    9E   D0    1678  4674          MOVL    @(SP)+,R4                ; Get IDB address
 55    18 A4   D0    167B  4675          MOVL    IDB$L_UCBLST(R4),R5      ; Get first UCB address
                     167F  4676          ASSUME  IDB$L_CSR EQ 0
    52    64   D0    167F  4677          MOVL    (R4),R2                  ; Get CSR address
 54    24 A5   D0    1682  4678          MOVL    UCB$L_CRB(R5),R4         ; Get CRB address
 54    10 A4   D0    1686  4679          MOVL    CRB$L_AUXSTRUC(R4),R4    ; Get CDB address
          2B   13    168A  4680          BEQL    INTEXIT                  ; Br if CDB not allocated
          00   E1    168C  4681          BBC     #CDB_STS_V_INITED,-      ; Br if NOT inited
 25 024A  C4         168E  4682                  CDB_B_STS(R4),INTEXIT    ;
          04   E0    1692  4683          BBS     #CDB_STS_V_ERR,-         ; Br if there was an error
 1F 024A  C4         1694  4684                  CDB_B_STS(R4),INTEXIT    ;
 53    0E A2   3C    1698  4685          MOVZWL  CSR(R2),R3               ; Fetch the CSR contents
                     169C  4686 ;
                     169C  4687 ; The interrupt bits can only be cleared by writing one's into them, therefore
                     169C  4688 ; we will write one's into all bits which already have one's.
                     169C  4689 ;
  0E A2   53    B0    169C  4690          MOVW    R3,CSR(R2)               ; Release interrupt interlocks
53    8080 8F   B3    16A0  4691          BITW    #XQ_CSR_M_XMTINT!-       ; Is this a valid interrupt?
                     16A5  4692                    XQ_CSR_M_RCVINT,R3     ;
          04   13    16A5  4693          BEQL    20$                      ; Br if no, indicate error
                     16A7  4694 ;
                     16A7  4695 ; We will now check for any errors.
                     16A7  4696 ;
 0A 53    02   E1    16A7  4697 10$:     BBC     #XQ_CSR_V_NXM,R3,30$     ; Br if no errors
                     16AB  4698 20$:     SETBIT  #XQ_CSR_V_ERR,R3         ; Set fatal error indicator
                     16AF  4699          SETBIT  #CDB_STS_V_ERR,-         ; Ignore futher interrupts
                     16AF  4700                  CDB_B_STS(R4)            ;
          0C   10    16B5  4701 30$:     BSBB    SCHED_FORK               ; Schedule a fork process
                     16B7  4702
                     16B7  4703 INTEXIT:                                 ; Exit interrupt
```

```
50   8E   7D   16B7  4704          MOVQ    (SP)+,R0                ; Restore Regs
52   8E   7D   16BA  4705          MOVQ    (SP)+,R2
54   8E   7D   16BD  4706          MOVQ    (SP)+,R4
                02   16C0  4707    REI                             ; Dismiss the interrupt
```

```
                        16C1  4709          .SBTTL  SCHED_FORK - SCHEDULE THE FORK PROCESS
                        16C1  4710          .SBTTL  SCHED_FORKC - SCHEDULE THE FORK PROCESS WITH R3 CLEAR
                        16C1  4711 ;++
                        16C1  4712 ; SCHED_FORK - Schedule the fork process
                        16C1  4713 ; SCHED_FORKC - Schedule the fork process with R3 clear
                        16C1  4714 ;
                        16C1  4715 ; Functional description:
                        16C1  4716 ;
                        16C1  4717 ; This routine is called to schedule the error and I/O completion fork process.
                        16C1  4718 ; The last controller CSR values are saved for examination.  If the
                        16C1  4719 ; fork process is already pending, only the last CSR values are saved if there
                        16C1  4720 ; was an error.
                        16C1  4721 ;
                        16C1  4722 ; Inputs:
                        16C1  4723 ;
                        16C1  4724 ;       R3 = Last CSR value
                        16C1  4725 ;       R4 = CDB address
                        16C1  4726 ;
                        16C1  4727 ;       IPL = DIPL or higher
                        16C1  4728 ;
                        16C1  4729 ; Outputs:
                        16C1  4730 ;
                        16C1  4731 ;       R3 is cleared if SCHED_FORKC entry.
                        16C1  4732 ;       R4 is destroyed.
                        16C1  4733 ;       R5 = CDB address
                        16C1  4734 ;
                        16C1  4735 ; If XQ_CSR_V_ERR is set in CSR, then the following is returned:
                        16C1  4736 ;
                        16C1  4737 ;       CDB_L_CSR(R4) = new CSR value
                        16C1  4738 ;
                        16C1  4739 ;--
                        16C1  4740
                        16C1  4741 SCHED_FORKC::                                ; Schedule fork process, clr R3
                  53    D4  16C1  4742          CLRL    R3                      ; No device error
                        16C3  4743 SCHED_FORK::                                 ; Schedule fork process
 0C 024A C4     02    E2  16C3  4744          BBSS    #CDB_STS_V_FORK_PEND,CDB_B_STS(R4),10$ ; Br if fork pending
                        16C9  4745          ASSUME  CDB_L_FQFL EQ 0
           55   54    D0  16C9  4746          MOVL    R4,R5                   ; Get CDB fork block address
             EF'AF    9F  16CC  4747          PUSHAB  B^FORK_PROC             ; Else, set address of fork process
       00000000'GF    17  16CF  4748          JMP     G^EXE$FORK              ; Schedule the fork and return
                        16D5  4749
        04 53   0E    E1  16D5  4750 10$:     BBC     #XQ_CSR_V_ERR,R3,20$ ; Br if not error
        10 A4   53    D0  16D9  4751          MOVL    R3,CDB_L_CSR(R4)        ; Save last CSR value
                      05  16DD  4752 20$:     RSB                             ; Return to caller
```

```
                        16DE  4754          .SBTTL  FORK_PROC - Error and completion fork process handling
                        16DE  4755 ;++
                        16DE  4756 ; FORK_PROC - Error and completion fork processing
                        16DE  4757 ;
                        16DE  4758 ; Functional description:
                        16DE  4759 ;
                        16DE  4760 ; This routine is called as a fork process to handle errors and all completions
                        16DE  4761 ; pending.
                        16DE  4762 ;
                        16DE  4763 ; Inputs:
                        16DE  4764 ;
                        16DE  4765 ;       R3 = Last CSR value
                        16DE  4766 ;       R4 = CDB address
                        16DE  4767 ;       R5 = CDB address
                        16DE  4768 ;
                        16DE  4769 ;       IPL = FIPL
                        16DE  4770 ;
                        16DE  4771 ; Outputs:
                        16DE  4772 ;
                        16DE  4773 ;       R0-R5 are destroyed.
                        16DE  4774 ;
                        16DE  4775 ;--
                  0868' 16DE  4776          .WORD   TIMEOUT-.                       ; Offset to timeout routine
                        16E0  4777 FORK_PROC::                                    ; Error/completion fork process
                        16E0  4778          CLRBIT  #CDB_STS_V_FORK_PEND,CDB_B_STS(R4) ; Clear fork process flag
      20 53    0E    E1  16E6  4779          BBC     #XQ_CSR_V_ERR,R3,10$    ; Br if not an error
                        16EA  4780          SETBIT  #XM$V_ERR_FATAL,CDB_L_DEVDEPEND(R4) ; Indicate fatal error
  0117 C4    53    90  16F0  4781          MOVB    R3,CDB_L_DEVDEPEND+3(R4) ; Save low byte of CSR
  55   0118 C4    D0  16F5  4782          MOVL    CDB_L_UCB0(R4),R5       ; Get UCB #0 address
       0082 C5    B6  16FA  4783          INCW    UCB$W_ERRCNT(R5)        ; Bump error counter
            08D5  31  16FE  4784          BRW     SHUTDOWN_QNA            ; Shutdown the QNA device
                        1701  4785
                        1701  4786 3$:     ;
                        1701  4787          ; Process receive errors
                        1701  4788          ;
            0293  30    1701  4789          BSBW    RCV_ERROR               ; Process receive error
            008C  31    1704  4790          BRW     25$                     ; Abort messages
                        1707  4791
            015A  31    1707  4792 7$:      BRW     60$                     ; Complete transmits
                        170A  4793 ;
                        170A  4794 ; Complete any TRANSMITS or RECEIVES
                        170A  4795 ;
                        170A  4796 10$:     PUSHQ   R6                      ; Save R6, R7
                        170D  4797
            0523  30    170D  4798 15$:     BSBW    ASSEM_PKTS              ; Assemble receive packets
          F4 50   E9    1710  4799          BLBC    R0,7$                   ; Br on error or none
                        1713  4800          INCC    CDB_L_DBRCTR(R4)        ; Count blocks received
  50    1A A2   3C    171D  4801          MOVZWL  CXB$W_BCNT(R2),R0       ; Get byte count
                        1721  4802          CNTR    R0,CDB_L_BRCCTR(R4),L   ; Count bytes received
    16 38 A2    E9    172D  4803          BLBC    CXB$G_R_DEST(R2),17$    ; Br if not multicast
                        1731  4804          INCC    CDB_L_MBLCTR(R4)        ; Count multicast blocks received
                        173B  4805          CNTR    R0,CDB_L_MBYCTR(R4),L   ; Count multicast bytes received
             0E   E0    1747  4806 17$:     BBS     #RCV_STS_V_ERR,-        ; Br if FATAL receive error
       B5 14 A2          1749  4807            CXB$W_R_STS(R2),3$            ;
             0D   E1    174C  4808          BBC     #RCV_STS_V_ESETUP,-     ; Br if NOT an ESETUP receive
       06 14 A2          174E  4809            CXB$W_R_STS(R2),20$           ;
             05   E4    1751  4810          BBSC    #CDB_STS_V_SETUP,-      ; Br if SETUP in progress and clear it
```

```
                     1753  4811                    CDB_B_STS(R4),25$       ;
 3C 024A C4          1753  4811                    CDB_B_STS(R4),25$       ;
 51    44 A2    3C   1757  4812 20$:     MOVZWL   CXB$W_R_PTYPE(R2),R1    ; Pick up protocol type from buffer
                     175B  4813
                     175B  4814             .IF DF  POINT
0660 8F    51    B1   175B  4815             CMPW    R1,#XQ_C_STPRO          ;% Is this the startup protocol?
           08    12   1760  4816             BNEQ    21$                     ;% Br if not
         0E92    30   1762  4817             BSBW    FIND_POINT_UCB          ;% Find the point to point user!
      14 50    E9   1765  4818             BLBC    R0,22$                  ;% Br if failure, assume regular user
           29    11   1768  4819             BRB     25$                     ;% Else done with MSG block
                     176A  4820             .ENDC
                     176A  4821
0260 8F    51    B1   176A  4822 21$:     CMPW    R1,#NI_CTR_PROTYP       ; Is this the Remote Console protocol?
           0B    12   176F  4823             BNEQ    22$                     ; Br if not
           09    91   1771  4824             CMPB    #NI_CTR_READ,-          ; Is this a read counters request?
        46 A2        1773  4825                     CXB$T_R_USERDAT(R2)     ;
           05    12   1775  4826             BNEQ    22$                     ; Br if not
         0615    30   1777  4827             BSBW    MOP_CTR_REQUEST         ; Else, process the request for counters
           91    11   177A  4828             BRB     15$
         1547    30   177C  4829 22$:     BSBW    MATCH_PROTYP            ; Try to match protocol type
      1F 50    E8   177F  4830             BLBS    R0,27$                  ; Br if success
55    0214 C4    D0   1782  4831 23$:     MOVL    CDB_L_PRMUSER(R4),R5    ; Try to get the PROMISCUOUS user
           60    12   1787  4832             BNEQ    34$                     ; Br if one found
                     1789  4833             INCC    CDB_W_UFDCTR(R4),W      ; Else, no such protocol type
        10 A2    DD   1793  4834 25$:     PUSHL   CXB$L_LINK(R2)          ; Save next in chain
         FCA7    30   1796  4835             BSBW    ADDRCVLIST              ; Add buffer to receive list
           52 8ED0    1799  4836             POPL    R2                      ; Restore next buffer
           F5    12   179C  4837             BNEQ    25$                     ; Loop if more
         FF6C    31   179E  4838             BRW     15$                     ; Look for next completed buffer
                     17A1  4839 ;
                     17A1  4840 ; If there is a promiscuous user, then copy the packet for the promiscuous
                     17A1  4841 ; user. There is a chance that the data received is not for the protocol
                     17A1  4842 ; type user just found, we will have to re-verify that the destination
                     17A1  4843 ; address is for our physical address. This is because if we are running
                     17A1  4844 ; promiscuous mode, then we will receive all packets, including those not
                     17A1  4845 ; intended for the protocol user.
                     17A1  4846 ;
                     17A1  4847 27$:     ASSUME  NMA$C_STATE_ON EQ 0
                     17A1  4848          ASSUME  NMA$C_STATE_OFF EQ 1
 23 024B C4    E8    17A1  4849          BLBS    CDB_B_PRM(R4),32$       ; Br if hardware is NOT in promiscuous mode
    10 38 A2    E8   17A6  4850          BLBS    CXB$G_R_DEST(R2),30$    ; Br if multicast address, this
                     17AA  4851                                          ;  will be checked later.
       38 A2    D1   17AA  4852          CMPL    CXB$G_R_DEST(R2),-      ; Is this packet for this protocol
    025A C4         17AD  4853                  CDB_G_PHYADR(R4)        ;  user?
           D0    12   17B0  4854          BNEQ    23$                     ; Br if not, don't copy packet
       3C A2    B1   17B2  4855          CMPW    CXB$G_R_DEST+4(R2),-    ; Are we sure?
    025E C4         17B5  4856                  CDB_G_PHYADR+4(R4)      ;
           C8    12   17B8  4857          BNEQ    23$                     ; Br if no, dont't copy packet
                     17BA  4858 ;
                     17BA  4859 ;        Copy the packet for the promiscuous user
                     17BA  4860 ;
           55    DD   17BA  4861 30$:     PUSHL   R5                      ; Save user's UCB address
55    0214 C4    D0   17BC  4862          MOVL    CDB_L_PRMUSER(R4),R5    ; Get PROMISCUOUS user's UCB address
           03    13   17C1  4863          BEQL    31$                     ; Br if none
         0280    30   17C3  4864          BSBW    COPY_RCV                ; Give buffer to promiscous user
           55 8ED0    17C6  4865 31$:     POPL    R5                      ; Restore user's UCB address
                     17C9  4866 ;
                     17C9  4867 ; If multicast address is the destination, then make sure that multicast
```

```
                     17C9  4868 ; address is in multicast address list for this unit.
                     17C9  4869 ;
          1C 38 A2   E9  17C9  4870 32$:    BLBC    CXB$G_R_DEST(R2),34$     ; Br if physical address
              146A   30  17CD  4871         BSBW    MATCH_MULTI              ; Try to match multicast address
           16 50     E8  17D0  4872         BLBS    R0,34$                   ; Br if success
                     17D3  4873 33$:    INCC    UCB$W_XQ_MNECTR(R5),W    ; Else, multicast not enabled
                     17DD  4874         INCC    CDB_W_UFDCTR(R4),W       ; Also added in unrecognized frame dest
                AA   11  17E7  4875         BRB     25$                      ; Release buffer
                     17E9  4876 ;
                     17E9  4877 ; If the user did not request data chaining, then check to make sure he gets
                     17E9  4878 ; no chained buffers... may be user of Internal IRPs!
                     17E9  4879 ;
              10 A2  D5  17E9  4880 34$:    TSTL    CXB$L_LINK(R2)           ; Is this a chained message?
                 05  13  17EC  4881         BEQL    36$                      ; Br if not
                     17EE  4882         ASSUME  NMA$C_STATE_ON EQ 0
                     17EE  4883         ASSUME  NMA$C_STATE_OFF EQ 1
       4A 00DC C5    E8  17EE  4884         BLBS    UCB$B_XQ_DCH(R5),45$     ; Br if chaining not allowed
                     17F3  4885 ;
                     17F3  4886 ; If there is a pending receive I/O request, complete it.
                     17F3  4887 ; Otherwise, queue the buffer and, if enabled, deliver attention AST.
                     17F3  4888 ;
       50    42 A5   3C  17F3  4889 36$:    MOVZWL  UCB$W_DEVBUFSIZ(R5),R0   ; Get size of user's max buffer
                     17F7  4890         ;&&
                     17F7  4891         ;&& the following code could cause problems for an altstart user,
                     17F7  4892         ;&& if the altstart user happens to receive a buffer which is 1 or 2
                     17F7  4893         ;&& bytes longer than they are capable of handling. Only if the
                     17F7  4894         ;&& protocol is not "padded", because the size check allows for
                     17F7  4895         ;&& 2 bytes of count to be subtracted from the message size.
                     17F7  4896         ;&&
                     17F7  4897 ;
                     17F7  4898 ; Check the size of the received buffer against what the user protocol can
                     17F7  4899 ; handle.
                     17F7  4900 ;
                  02 A3  17F7  4901         SUBW3   #XQ_C_CNTSIZ,-           ; Get the size of the receive buffer
          51    1A A2    17F9  4902                 CXB$W_BCNT(R2),R1        ;  minus the count word
             50    51 B1  17FC  4903         CMPW    R1,R0                    ; Is the received size larger than
                     17FF  4904                                             ;  what the user can handle?
                  3C 1A  17FF  4905         BGTRU   45$                      ; Br if yes, error
              018D C5 D5  1801  4906         TSTL    UCB$L_XQ_FFI(R5)         ; FAST Interface supported?
                  06 13  1805  4907         BEQL    37$                      ; Br if not, standard interface
                02CE 30  1807  4908         BSBW    FINISH_RCV_FFI           ; Else, complete FAST receive
                FF00 31  180A  4909         BRW     15$                      ; Look for more completions
                     180D  4910
       51    00A8 C5 9E  180D  4911 37$:    MOVAB   UCB$Q_XQ_RCVREQ(R5),R1   ; Assume that we are running in
                     1812  4912                                             ;  exclusive mode
                  03 E1  1812  4913         BBC     #UCB$V_XQ_SHARE,-        ; Br if UCB is NOT in SHARED mode
             09 68 A5    1814  4914                 UCB$W_DEVSTS(R5),38$     ;
                     1817  4915 ;
                     1817  4916 ; If running in SHARED mode, then we must use the listheads in the SHR
                     1817  4917 ; data structure. We will use the source address from the received message
                     1817  4918 ; to match against the SHR structure destination address.
                     1817  4919 ;
                01F8 30  1817  4920         BSBW    MATCH_SRC                ; Check for a match on source address
                  B7 12  181A  4921         BNEQ    33$                      ; Br if no shared user found
                     181C  4922 ;
                     181C  4923 ;       SHARED user found, use listheads in SHR data structure.
                     181C  4924 ;
```

```
                51    20 A1    9E  181C  4925                MOVAB   SHR_Q_RCVREQ(R1),R1      ; Get address of waiting IRPs
                53    00 B1    0F  1820  4926 38$:          REMQUE  @(R1),R3                ; Remove waiting IRP
                         0B    1D  1824  4927                BVS     40$                     ; Br if none - queue for later
              00C8 C5    50    A2  1826  4928                SUBW    R0,UCB$W_XQ_QUOTA(R5)   ; Else, lessen quota so it can be
                                   182B  4929                                                ;..increased on completion
                      02E5    30  182B  4930                BSBW    FINISH_RCV_IO           ; And finish the I/O
                      FEDC    31  182E  4931                BRW     15$                     ; Look for next completion
                                  1831  4932 ;
                                  1831  4933 ; Check buffer quota and queue if quota okay.
                                  1831  4934 ;
              00C8 C5    50    A2  1831  4935 40$:          SUBW    R0,UCB$W_XQ_QUOTA(R5)   ; Decrement the quota
                         1C    1E  1836  4936                BGEQU   50$                     ; Br if we can buffer request
              00C8 C5    50    A0  1838  4937                ADDW    R0,UCB$W_XQ_QUOTA(R5)   ; Replace quota
                                  183D  4938 45$:          INCC    UCB$W_XQ_UBUCTR(R5),W   ; Else, no buffer available
                                  1847  4939                INCC    CDB_W_UBUCTR(R4),W      ; ...don't forget CDB counter
                      FF3F    31  1851  4940                BRW     25$                     ; Return buffer
                                  1854  4941
                                  1854  4942 50$:          ASSUME  UCB$Q_XQ_RCVREQ EQ UCB$Q_XQ_RCVMSG+8
                                  1854  4943                ASSUME  SHR_Q_RCVREQ EQ SHR_Q_RCVMSG+8
                   51    04    C2  1854  4944                SUBL    #4,R1                   ; Backup to backward link pointer
                                  1857  4945                                                ;  of the message queue
                00 B1    62    0E  1857  4946                INSQUE  (R2),@(R1)              ; Queue received msg for later
                      FBE0    30  185B  4947                BSBW    FILLRCVLIST             ; Try to fill the receive list
                      1430    30  185E  4948                BSBW    POKE_USER               ; Deliver ASTs
                      FEA9    31  1861  4949                BRW     15$                     ; Look for more completions
                                  1864  4950 ;
                                  1864  4951 ; NOW - scan the xmit ring entries
                                  1864  4952 ;
                   010F C4    95  1864  4953 60$:          TSTB    CDB_B_XMTCNT(R4)        ; Any xmits in progress?
                         03    12  1868  4954                BNEQ    70$                     ; Br if yes - look for any completed
                      0126    31  186A  4955 65$:          BRW     190$                    ; Else, all done
                                  186D  4956
              56    010D C4    9A  186D  4957 70$:          MOVZBL  CDB_B_LASTXMT(R4),R6    ; Get last ring entry completed
            56   009C C446    D0  1872  4958                MOVL    CDB_L_XRINGVA(R4)[R6],R6; Get address of last ring entry
                         0F    E1  1878  4959                BBC     #XMT_STS_V_LAST,-       ; Br if done
                05 08 A6                  187A  4960                 XMT_W_STS(R6),75$      ;
                         0E    E0  187D  4961                BBS     #XMT_STS_V_ERR,-        ; Br if not done
                E8 08 A6                  187F  4962                 XMT_W_STS(R6),65$      ;  ..leave
                   0A A6    B5  1882  4963 75$:          TSTW    XMT_Q_TDR(R6)           ; Are we really done?
                         E3    13  1885  4964                BEQL    65$                     ; Br if not!
              53    00EC D4    0F  1887  4965                REMQUE  @CDB_Q_XMTPND(R4),R3    ; Get next XMIT CXB
                         DC    1D  188C  4966                BVS     65$                     ; Br if none there (yet)
                         04    12  188E  4967                BNEQ    77$                     ; Br if more entries on queue
                   020E C4    94  1890  4968                CLRB    CDB_B_TIM_XMT(R4)       ; Stop the xmit timer
                   010F C4    97  1894  4969 77$:          DECB    CDB_B_XMTCNT(R4)        ; One less transmit pending
                   010D C4    96  1898  4970                INCB    CDB_B_LASTXMT(R4)       ; Bump ring pointer
                      FC 8F    8A  189C  4971                BICB    #^C<MAX_C_XMT-1>,-      ; Modulo receive ring entry size
                   010D C4        189F  4972                         CDB_B_LASTXMT(R4)      ;       ...
                                  18A2  4973 ;
                                  18A2  4974 ; Transmit complete
                                  18A2  4975 ;
                                  18A2  4976                CLRBIT  XMT_DSC_V_VALID,-       ; Indicate that buffer is not valid
                                  18A2  4977                         XMT_W_ADDRHI(R6)       ;
                      01 A6    90  18A7  4978                MOVB    XMT_Q_FLAG+1(R6),-      ; Save diagnostic return info
                   011D C4        18AA  4979                         CDB_B_DIAG1(R4)        ;
                      08 A6    B0  18AD  4980                MOVW    XMT_Q_STS(R6),-         ; Save diagnostic return info
                   011E C4        18B0  4981                         CDB_Q_DIAG2(R4)        ;
```

```
                55   24 A3   D0  18B3  4982          MOVL    CXB$L_T_UCB(R3),R5      ; Get (presumed) UCB address
                                 18B7  4983          ASSUME  CXB$L_T_UCB  EQ  CXB$L_T_IRP
                04 55   00   E4  18B7  4984          BBSC    #0,R5,80$               ; Fix up address & BR if UCB address
                55   1C A5   D0  18BB  4985          MOVL    IRP$L_UCB(R5),R5        ; Else, get real UCB address
                                 18BF  4986
                52   22 A3   9A  18BF  4987 80$:     MOVZBL  CXB$B_XQ_SLOT(R3),R2    ; Get mapping slot number used
                                 18C3  4988          CLRBIT  R2,CDB_B_XMTMAP(R4)     ; Clear in use flag
                                 18C8  4989
                                 18C8  4990          CPUDISP <<790,90$>,-
                                 18C8  4991                   <780,90$>,-
                                 18C8  4992                   <750,90$>,-
                                 18C8  4993                   <730,90$>,-
                                 18C8  4994                   <UV1,100$>>            ; Skip map registers if u-VAX I
                                 18E2  4995
                51   24 A5   D0  18E2  4996 90$:     MOVL    UCB$L_CRB(R5),R1        ; Get CRB address
                                 18E6  4997          ASSUME  VEC$B_NUMREG EQ VEC$W_MAPREG+2
                                 18E6  4998          ASSUME  VEC$B_DATAPATH EQ VEC$B_NUMREG+1
          34 A1   38 A442   D0  18E6  4999          MOVL    CDB_L_XMTMAP(R4)[R2],-  ; Setup map register data
                                 18EC  5000                   CRB$L_INTD+VEC$W_MAPREG(R1) ;  and data path number
                      52   95  18EC  5001          TSTB    R2                      ; Was it the pre-allocated one?
                      0D   13  18EE  5002          BEQL    95$                     ; Br if yes - clear data path number
             38 A442    01   CE  18F0  5003          MNEGL   #1,CDB_L_XMTMAP(R4)[R2] ; Indicate map register not allocated
                                 18F5  5004          RELMPR                          ; Release the map registers
                      03   11  18FB  5005          BRB     100$                    ; Complete the request
                                 18FD  5006
                                 18FD  5007 95$:     ASSUME  VEC$B_DATAPATH EQ VEC$W_MAPREG+3
                   3B A4   94  18FD  5008          CLRB    CDB_L_XMTMAP+3(R4)      ; Clear data path number used
                50   1A A3   3C  1900  5009 100$:    MOVZWL  CXB$W_BCNT(R3),R0       ; Get byte count of message
                   51   50   D0  1904  5010          MOVL    R0,R1                   ; Copy length for accounting
                                 1907  5011          ;
                                 1907  5012          ; Perform accounting for the QNA
                                 1907  5013          ;
                                 1907  5014          INCC    CDB_L_DBSCTR(R4)        ; Count blocks sent
                                 1911  5015          CNTR    R1,CDB_L_BSNCTR(R4),L   ; Count bytes sent
             50   50   10   78  191D  5016          ASHL    #16,R0,R0               ; Move to high word
                50   01   B0  1921  5017          MOVW    S^#SS$_NORMAL,R0        ; Set completion status
                     0C   E4  1924  5018          BBSC    #XMT_DSC_V_SETUP,-      ; Br if SETUP operation
                   42 02 A6      1926  5019                  XMT_W_ADDRAI(R6),120$   ;
          05 08 A6     0E   E1  1929  5020          BBC     #XMT_STS_V_ERR,XMT_W_STS(R6),110$ ; Br if not a FATAL error
                      0092   30  192E  5021          BSBW    XMT_ERROR               ; Process XMIT error
                      38   11  1931  5022          BRB     120$                    ; Skip the accounting
                                 1933  5023 ;
                                 1933  5024 ; Perform accounting on a per protocol type basis and on unit.
                                 1933  5025 ;
                                 1933  5026 110$:    CNTR    R1,UCB$L_XQ_SBYCTR(R5),L ; Bump the bytes sent counter
                                 193F  5027          INCC    UCB$L_XQ_SBDCTR(R5),L   ; Bump the blocks sent counter
                      04   EF  1949  5028          EXTZV   #XMT_STS_V_COL,-        ; Get number of collisions
          51   08 A6     04      194B  5029                  #XMT_STS_S_COL,XMT_W_STS(R6),R1
                      1A   13  194F  5030          BEQL    120$                    ; Br if none
                      51   97  1951  5031          DECB    R1                      ; More than one?
                      0C   13  1953  5032          BEQL    115$                    ; Br if only one
                                 1955  5033          INCC    CDB_L_BSMCTR(R4)        ; Count blocks sent with multiple errors
                      0A   11  195F  5034          BRB     120$                    ; Continue
                                 1961  5035 115$:    INCC    CDB_L_BS1CTR(R4)        ; Count blocks sent with 1 error
                                 196B  5036
                1B 24 A3   E8  196B  5037 120$:    BLBS    CXB$L_T_IRP(R3),130$    ; BR if FAST interface CXB
             53   24 A3   D0  196F  5038          MOVL    CXB$L_T_IRP(R3),R3      ; Else, get IRP address
```

```
                50    DD  1973  5039          PUSHL   R0                              ; Save status return
50   0094      C3    D0  1975  5040          MOVL    IRP$L_XQ_SETUP(R3),R0           ; Is there a SETUP mode buffer
                06    13  197A  5041          BEQL    125$                            ; Br if none
00000000'       GF    16  197C  5042          JSB     G^COM$DRVDEALMEM                ; Deallocate the buffer
                50  8ED0  1982  5043 125$:    POPL    R0                              ; Restore status return
              0254    30  1985  5044          BSBW    IO_DONE                         ; Finish the I/O request
                03    11  1988  5045          BRB     170$                            ; Continue in common code
              013D    30  198A  5046 130$:    BSBW    FINISH_XMT_FFI                  ; Else, complete FFI XMIT
              EB03    30  198D  5047 170$:    BSBW    XMT_ALT_START                   ; Start up any other transmits
              FED1    31  1990  5048          BRW     60$                             ; Look for next completion
                      1993  5049
                      1993  5050 190$:    POPQ    R6                              ; Restore R6, R7
                05    1996  5051          RSB
                      1997  5052
```

```
                     1997  5054          .SBTTL  RCV_ERROR - Process receive errors
                     1997  5055          .SBTTL  XMT_ERROR - Process transmit errors
                     1997  5056 ;++
                     1997  5057 ; RCV_ERROR - Process receive errors
                     1997  5058 ;
                     1997  5059 ; Functional description:
                     1997  5060 ;
                     1997  5061 ; This routine adjusts all appropriate counters and checks all errors.
                     1997  5062 ;
                     1997  5063 ; Inputs:
                     1997  5064 ;       R2 = CXB address
                     1997  5065 ;       R4 = CDB address
                     1997  5066 ;
                     1997  5067 ; Outputs:
                     1997  5068 ;       none.
                     1997  5069 ;
                     1997  5070 ;--
                     1997  5071
                     1997  5072 RCV_ERROR:
                     1997  5073          INCC    CDB_W_RFLCTR(R4),W      ; Count receive failures
          01   E1    19A1  5074          BBC     #RCV_STS_V_CRCERR,-     ; Br if not a CRC error
06 14 A2            19A3  5075                  CXB$W_R_STS(R2),20$     ;
                     19A6  5076          SETBIT  #0,CDB_W_RFLMAP(R4)     ; Indicate CRC error
          02   E1    19AC  5077 20$:     BBC     #RCV_STS_V_FRAME,-      ; Br if not a framing error
06 14 A2            19AE  5078                  CXB$W_R_STS(R2),40$     ;
                     19B1  5079          SETBIT  #1,CDB_W_RFLMAP(R4)     ; Indicate FRAME error
          0B   E1    19B7  5080 40$:     BBC     #RCV_STS_V_RUNT,-       ; Br if not a RUNT packet
06 14 A2            19B9  5081                  CXB$W_R_STS(R2),90$     ;
                     19BC  5082          SETBIT  #2,CDB_W_RFLMAP(R4)     ; Indicate FRAME error
          05         19C2  5083 90$:     RSB
                     19C3  5084
                     19C3  5085
                     19C3  5086 ;++
                     19C3  5087 ; XMT_ERROR - Process transmit errors
                     19C3  5088 ;
                     19C3  5089 ; Functional description:
                     19C3  5090 ;
                     19C3  5091 ; This routine adjusts all appropriate counters and checks all errors.
                     19C3  5092 ;
                     19C3  5093 ; Inputs:
                     19C3  5094 ;       R4 = CDB address
                     19C3  5095 ;       R6 = Transmit ring entry address
                     19C3  5096 ;
                     19C3  5097 ; Outputs:
                     19C3  5098 ;       R0 = error code
                     19C3  5099 ;
                     19C3  5100 ;--
                     19C3  5101
                     19C3  5102 XMT_ERROR:
                     19C3  5103          INCC    CDB_W_SFLCTR(R4),W      ; Count send failures
50   20C4 8F    B0   19CD  5104          MOVW    #SS$_COMMHARD,R0        ; Assume No Carrier failure
          09    E1   19D2  5105          BBC     #XMT_STS_V_ABORT,-      ; Br if NOT 16 retries failed
     0B 08 A6         19D4  5106                  XMT_W_STS(R6),20$
50   0334 8F    B0   19D7  5107          MOVW    #SS$_DEVREQERR,R0       ; Else, DEVREQERR error
                     19DC  5108          SETBIT  #0,CDB_W_SFLMAP(R4)     ; Set bitmap
          0C    E1   19E2  5109 20$:     BBC     #XMT_STS_V_LCAR,-       ; Br if NOT Loss of Carrier
     0B 08 A6         19E4  5110                  XMT_W_STS(R6),40$
```

```
                50   204C 8F    B0  19E7  5111          MOVW    #SS$_DISCONNECT,R0      ; Else, DISCONNECT error
                                    19EC  5112          SETBIT  #1,CDB_W_SFLMAP(R4)     ; Set bitmap
                          08    E1  19F2  5113 40$:     BBC     #XMT_STS_V_FAIL,-       ; Br if NOT Collision check failure
                     0F 08 A6        19F4  5114                  XMT_W_STS(R6),60$
                50   005C 8F    B0  19F7  5115          MOVW    #SS$_DATACHECK,R0       ; Else, DATACHECK error
                                    19FC  5116          INCC    CDB_W_CDCCTR(R4),W      ; Count separate counter
                          0B    E1  1A06  5117 60$:     BBC     #XMT_STS_V_NOCAR,-      ; Br if NOT Carrier failure
                     06 08 A6        1A08  5118                  XMT_W_STS(R6),90$       ;
                                     1A0B  5119          SETBIT  #2,CDB_W_SFLMAP(R4)     ; Set bitmap
                                05   1A11  5120 90$:     RSB
```

```
                                1A12  5122          .SBTTL   SUBROUTINES TO FIND SHR MATCH ON SOURCE ADDRESS
                                1A12  5123 ;++
                                1A12  5124 ; Functional description:
                                1A12  5125 ;
                                1A12  5126 ; Subroutine to find SHR data structure for user
                                1A12  5127 ;
                                1A12  5128 ; Inputs:
                                1A12  5129 ;       R2 = Receive CXB address
                                1A12  5130 ;       R5 = UCB address
                                1A12  5131 ;
                                1A12  5132 ; Outputs:
                                1A12  5133 ;       R1 = Address if SHR data structure if match
                                1A12  5134 ;       All other registers preserved.
                                1A12  5135 ;       Z-Bit set then match.
                                1A12  5136 ;       Z-Bit clear then no match.
                                1A12  5137 ;--
                                1A12  5138
                                1A12  5139 MATCH_SRC:                                   ; Try to find shared user
                     05     BB  1A12  5140          PUSHR   #^M<R0,R2>                 ; Save registers
                                1A14  5141 ;
                                1A14  5142 ; Try to find match among limited shared users of protocol type
                                1A14  5143 ;
                50   0098 C5  9E  1A14  5144          MOVAB   UCB$Q_XQ_SHARE(R5),R0      ; Save address of listhead
                     51   50  D0  1A19  5145          MOVL    R0,R1                      ; Copy listhead address
                                1A1C  5146          ASSUME  SHR_L_QFL EQ 0
                     51   61  D0  1A1C  5147 20$:    MOVL    (R1),R1                    ; Get next in list
                     50   51  D1  1A1F  5148          CMPL    R1,R0                      ; Back to start of list?
                          06  13  1A22  5149          BEQL    30$                        ; Br if yes - no source match
                          13  10  1A24  5150          BSBB    CHECK_SRC                  ; Check for match
                          F4  12  1A26  5151          BNEQ    20$                        ; Br if none
                          0C  11  1A28  5152          BRB     50$                        ; Return in success (Z-bit is set)
                                1A2A  5153 ;
                                1A2A  5154 ; No match on limited users - try to use default user
                                1A2A  5155 ;
              51   00C4 C5  D0  1A2A  5156 30$:    MOVL    UCB$L_XQ_DEFUSR(R5),R1     ; Get address of default user
                          02  13  1A2F  5157          BEQL    40$                        ; Br if no default user
                          50  D4  1A31  5158          CLRL    R0                         ; Return success
                     50   50  D0  1A33  5159 40$:    MOVL    R0,R0                      ; Return success/failure indicator
                          05  BA  1A36  5160 50$:    POPR    #^M<R0,R2>                 ; Restore registers, don't reset Z-BIT
                              05  1A38  5161          RSB
                                1A39  5162
                                1A39  5163 ;++
                                1A39  5164 ; Functional description:
                                1A39  5165 ;
                                1A39  5166 ; Subroutine to check if source address in message matches SHR address
                                1A39  5167 ;
                                1A39  5168 ; Inputs:
                                1A39  5169 ;       R1 = Address of SHR
                                1A39  5170 ;       R2 = Address of MSG buffer
                                1A39  5171 ;
                                1A39  5172 ; Outputs:
                                1A39  5173 ;       Z-Bit set then match.
                                1A39  5174 ;       Z-Bit clear then no match.
                                1A39  5175 ;--
                                1A39  5176
                                1A39  5177 CHECK_SRC:                                   ; Check for match with SHR data base
          12 A1   3E A2  D1     1A39  5178          CMPL    CXB$G_R_SRC(R2),SHR_G_DEST(R1) ; Source address match?
```

```
                05   12  1A3E  5179          BNEQ    10$                             ; Br if no - try for next
            42 A2   B1  1A40  5180          CMPW    CXB$G_R_SRC+4(R2),-             ; Really match?
            16 A1       1A43  5181           SHR_G_DEST+4(R1)                       ;          ...
                05   1A45  5182 10$:        RSB                                     ; Return to caller
```

```
                        1A46  5184          .SBTTL  COPY_RCV - Copy a receive buffer for the PROMISCUOUS user
                        1A46  5185 ;++
                        1A46  5186 ; COPY_RCV - Copy a receive buffer for the PROMISCUOUS user
                        1A46  5187 ;
                        1A46  5188 ; Functional description:
                        1A46  5189 ;
                        1A46  5190 ; This routine allocates a receive buffer in which to copy the a receive
                        1A46  5191 ; buffer for the PROMISCUOUS user.
                        1A46  5192 ;
                        1A46  5193 ;  Inputs:
                        1A46  5194 ;
                        1A46  5195 ;       R2 = Receive CXB address
                        1A46  5196 ;       R4 = CDB address
                        1A46  5197 ;       R5 = UCB address of PROMISCUOUS user
                        1A46  5198 ;       R6 = Address of receive ring entry
                        1A46  5199 ;
                        1A46  5200 ;       IPL = FIPL
                        1A46  5201 ;
                        1A46  5202 ;  Outputs:
                        1A46  5203 ;
                        1A46  5204 ;       R0,R1,R3 are destroyed.
                        1A46  5205 ;       All other registers are preserved.
                        1A46  5206 ;--
                        1A46  5207 COPY_RCV:                                    ; Copy the xmit buffer to rcv buffer
              52   DD   1A46  5208          PUSHL   R2                        ; Save address of original buffer
        1C A2      01   B1   1A48  5209          CMPW    #1,CXB$W_R_NCHAIN(R2)     ; Is there more than 1 in chain?
              78   12   1A4C  5210          BNEQ    80$                       ; Br if yes - we can only handle 1
        51    1A A2   3C   1A4E  5211          MOVZWL  CXB$W_BCNT(R2),R1         ; Get size of buffer
51      00000048 8F   C0   1A52  5212          ADDL    #CXB$C_HEADER,R1          ; Compute size of needed buffer
        00000000'GF   16   1A59  5213          JSB     G^EXE$ALONONPAGED         ; Allocate a receive buffer
           64 50      E9   1A5F  5214          BLBC    R0,80$                    ; Br if failure
        53    6E      D0   1A62  5215          MOVL    (SP),R3                   ; Get address of original buffer
              36      BB   1A65  5216          PUSHR   #^M<R1,R2,R4,R5>          ; Save registers
62    63      51      28   1A67  5217          MOVC3   R1,(R3),(R2)              ; Copy everything to new buffer
              36      BA   1A6B  5218          POPR    #^M<R1,R2,R4,R5>          ; Save registers
  08 A2       51      B0   1A6D  5219          MOVW    R1,CXB$W_SIZE(R2)         ; Reset size field
                           1A71  5220 ;
                           1A71  5221 ; If there is a pending receive I/O request, complete it.
                           1A71  5222 ; Otherwise, queue the buffer and, if enabled, deliver attention AST.
                           1A71  5223 ;
  51    42 A5       3C   1A71  5224          MOVZWL  UCB$W_DEVBUFSIZ(R5),R1    ; Get the user's buffer size
53    00A8 D5       0F   1A75  5225          REMQUE  @UCB$Q_XQ_RCVREQ(R5),R3   ; Remove waiting IRP
              1D      1D   1A7A  5226          BVS     40$                       ; Br if none - queue for later
00C8 C5       51      A2   1A7C  5227          SUBW    R1,UCB$W_XQ_QUOTA(R5)     ; Else, lessen quota so it can be
                           1A81  5228                                            ;..increased on completion
          008F      30   1A81  5229          BSBW    FINISH_RCV_IO             ; And finish the I/O
            40      11   1A84  5230          BRB     80$                       ; Look for next completion
                           1A86  5231 ;
                           1A86  5232 ; Return buffer to pool if user buffer failure
                           1A86  5233 ;
        10 A2       DD   1A86  5234 30$:     PUSHL   CXB$L_LINK(R2)            ; Save next in chain
     50    52       D0   1A89  5235          MOVL    R2,R0                     ; Copy the buffer address
00000000'GF         16   1A8C  5236          JSB     G^COM$DRVDEALMEM          ; DEALLOCATE the buffer
              52 8ED0   1A92  5237          POPL    R2                        ; Restore next buffer
              EF      12   1A95  5238          BNEQ    30$                       ; Loop if more
              2D      11   1A97  5239          BRB     80$                       ; Else, leave
                           1A99  5240 ;
```

```
                        1A99  5241 ; Check buffer quota and queue if quota okay.
                        1A99  5242 ;
00C8 C5   51    A2      1A99  5243 40$:    SUBW    R1,UCB$W_XQ_QUOTA(R5)    ; Decrement the quota
          1B    1E      1A9E  5244         BGEQU   50$                      ; Br if we can buffer request
00C8 C5   51    A0      1AA0  5245         ADDW    R1,UCB$W_XQ_QUOTA(R5)    ; Replace quota
                        1AA5  5246         INCC    UCB$W_XQ_UBOCTR(R5),W    ; Else, no buffer available
                        1AAF  5247         INCC    CDB_W_UBOCTR(R4),W       ; ...don't forget CDB counter
          CB    11      1AB9  5248         BRB     30$                      ; Return buffer
                        1ABB  5249
00A4 D5   62    0E      1ABB  5250 50$:    INSQUE  (R2),@UCB$Q_XQ_RCVMSG+4(R5) ; Queue received msg for later
        F97B    30      1AC0  5251         BSBW    FILLRCVLIST              ; Try to fill the receive list
        11CB    30      1AC3  5252         BSBW    POKE_USER                ; Deliver ASTs
          52  8ED0      1AC6  5253 80$:    POPL    R2                       ; Restore R2
              05        1AC9  5254         RSB                              ; Return to caller
```

```
                     1ACA  5256          .SBTTL  FINISH_XMT_FFI - Finish FAST interface transmit processing
                     1ACA  5257 ;++
                     1ACA  5258 ; FINISH_XMT_FFI - Finish FAST interface transmit processing
                     1ACA  5259 ;
                     1ACA  5260 ; Functional description:
                     1ACA  5261 ;
                     1ACA  5262 ; This routine completes a transmit CXB for a particular user of the fast
                     1ACA  5263 ; interface.
                     1ACA  5264 ;
                     1ACA  5265 ;  Inputs:
                     1ACA  5266 ;
                     1ACA  5267 ;       R0 = Status of transmit request
                     1ACA  5268 ;       R3 = transmit CXB address
                     1ACA  5269 ;       R4 = CDB address
                     1ACA  5270 ;       R5 = UCB address
                     1ACA  5271 ;
                     1ACA  5272 ;       IPL = FIPL
                     1ACA  5273 ;
                     1ACA  5274 ;  Outputs:
                     1ACA  5275 ;
                     1ACA  5276 ;       R0-R3 are destroyed.
                     1ACA  5277 ;       All other registers are preserved.
                     1ACA  5278 ;
                     1ACA  5279 ;--
                     1ACA  5280          ASSUME  IPL$_SYNCH EQ IPL$_XQ_FIPL
                     1ACA  5281 FINISH_XMT_FFI::                         ; Finish FAST interface transmit request
           54    DD  1ACA  5282          PUSHL   R4                      ; Save R4
54   018D C5    D0  1ACC  5283          MOVL    UCB$L_XQ_FFI(R5),R4     ; Get FFI block address
     14 B4      16  1AD1  5284          JSB     @FFI$L_XMIT_DONE(R4)    ; Call back the user with CXB
        54 8ED0     1AD4  5285          POPL    R4                      ; Restore R4
           05       1AD7  5286          RSB
```

```
                                1AD8  5288          .SBTTL  FINISH_RCV_FFI - Finish FAST receive processing
                                1AD8  5289 ;++
                                1AD8  5290 ; FINISH_RCV_FFI - Finish FAST receive processing
                                1AD8  5291 ;
                                1AD8  5292 ; Functional description:
                                1AD8  5293 ;
                                1AD8  5294 ; This routine completes a receive CXB for a particular user of the fast
                                1AD8  5295 ; interface.
                                1AD8  5296 ;
                                1AD8  5297 ;  Inputs:
                                1AD8  5298 ;
                                1AD8  5299 ;       R2 = receive CXB address
                                1AD8  5300 ;       R4 = CDB address
                                1AD8  5301 ;       R5 = UCB address
                                1AD8  5302 ;
                                1AD8  5303 ;       IPL = FIPL
                                1AD8  5304 ;
                                1AD8  5305 ;  Outputs:
                                1AD8  5306 ;
                                1AD8  5307 ;       R0-R3 are destroyed.
                                1AD8  5308 ;       All other registers are preserved.
                                1AD8  5309 ;
                                1AD8  5310 ;--
                                1AD8  5311          ASSUME  IPL$_SYNCH EQ IPL$_XQ_FIPL
                                1AD8  5312 FINISH_RCV_FFI::                         ; Finish FAST recieve request
                          54    DD  1AD8  5313          PUSHL   R4                  ; Save R4
                      53  52    D0  1ADA  5314          MOVL    R2,R3               ; Copy CXB address
                        3E  A3   7D  1ADD  5315          MOVQ    CXB$G_R_SRC(R3),-   ; Copy source address
                        28  A3       1AE0  5316                  CXB$Q_STATION(R3)   ;                    ...
                18  A3  0046  8F    B0  1AE2  5317          MOVW    #CXB$T_R_USERDAT,CXB$W_BOFF(R3) ; Set offset to received data
                                1AE8  5318          ASSUME  NMA$C_STATE_ON EQ 0
                                1AE8  5319          ASSUME  NMA$C_STATE_OFF EQ 1
                    09  00D9  C5    E8  1AE8  5320          BLBS    UCB$B_XQ_PAD(R5),30$    ; Br if padding is disabled
                          46  A3    B0  1AED  5321          MOVW    CXB$W_R_SIZE(R3),-      ; Else, set real size of buffer
                          1A  A3        1AF0  5322                  CXB$W_BCNT(R3)
                    18  A3    02    A0  1AF2  5323          ADDW    #XQ_C_CNTSIZ,CXB$W_BOFF(R3) ; Adjust offset
                50  0114  C4    D0  1AF6  5324  30$:     MOVL    CDB_L_DEVDEPEND(R4),R0  ; Set controller bits
                  50    44  A5    C8  1AFB  5325          BISL    UCB$L_DEVDEPEND(R5),R0  ; Set status flags
                54  018D  C5    D0  1AFF  5326          MOVL    UCB$L_XQ_FFI(R5),R4     ; Get FFI block address
                          18  B4    16  1B04  5327          JSB     @FFI$L_RECV_DONE(R4)    ; Call back the user with CXB
                              54  8ED0  1B07  5328          POPL    R4                      ; Restore R4
                          52    53    D0  1B0A  5329          MOVL    R3,R2                   ; Was buffer consumed?
                                03    13  1B0D  5330          BEQL    90$                     ; Br if YES
                                  F92E    30  1B0F  5331          BSBW    ADDRCVLIST              ; Else, add buffer to receive list
                                        05  1B12  5332  90$:     RSB
```

```
                        1B13  5334          .SBTTL  FINISH_RCV_IO - Finish receive I/O processing
                        1B13  5335 ;++
                        1B13  5336 ; FINISH_RCV_IO - Finish receive I/O processing
                        1B13  5337 ;
                        1B13  5338 ; Functional description:
                        1B13  5339 ;
                        1B13  5340 ; This routine completes a receive operation that has been matched with a
                        1B13  5341 ; message block. After the receive has been completed the message free list
                        1B13  5342 ; is filled and a receive is started if needed.
                        1B13  5343 ;
                        1B13  5344 ;  Inputs:
                        1B13  5345 ;
                        1B13  5346 ;       R2 = receive CXB address
                        1B13  5347 ;       R3 = I/O packet address
                        1B13  5348 ;       R4 = CDB address
                        1B13  5349 ;       R5 = UCB address
                        1B13  5350 ;
                        1B13  5351 ;       IPL = FIPL
                        1B13  5352 ;
                        1B13  5353 ;  Outputs:
                        1B13  5354 ;
                        1B13  5355 ;       R5 is reset to UCB address from IRP
                        1B13  5356 ;
                        1B13  5357 ;       The request is completed via I/O post.
                        1B13  5358 ;--
                        1B13  5359
                        1B13  5360 FINISH_RCV_IO::                          ; Finish recieve I/O request
2C A3    52    D0       1B13  5361          MOVL    R2,IRP$L_SVAPTE(R3)    ; Save block address
     3E A2    7D       1B17  5362          MOVQ    CXB$G_R_SRC(R2),-      ; Copy source address for DECnet
     40 A3             1B1A  5363                  IRP$Q_STATION(R3)      ;       ...
        07    E1       1B1C  5364          BBC     #IRP$V_DIAGBUF,-       ; Br if no diagnotic buffer
          27 2A A3      1B1E  5365                  IRP$W_STS(R3),10$      ;
        51    4C A3    D0    1B21  5366          MOVL    IRP$L_DIAGBUF(R3),R1   ; Get diagnostic buffer
        50    0C A1    9E    1B25  5367          MOVAB   RHDR_T_DATA(R1),R0     ; Assume this is just a read header
        08 A1          1A    B1    1B29  5368          CMPW    #RHDR_C_LENGTH,DIAG_W_SIZE(R1) ; Is this just a header buffer?
                       10    13    1B2D  5369          BEQL    5$             ; Br if yes
        50    30 A1    9E    1B2F  5370          MOVAB   DIAG_T_RDATA(R1),R0    ; Else, must be a diagnostic buffer
              0B A2    90    1B33  5371          MOVB    CXB$B_R_FLAGS(R2),-    ; Save diagnostic return info
           011D C4          1B36  5372                  CDB_B_DIAG1(R4)        ;       ...
              14 A2    B0    1B39  5373          MOVW    CXB$W_R_STS(R2),-      ;       ...
           011E C4          1B3C  5374                  CDB_W_DIAG2(R4)        ;       ...
                    3C    BB    1B3F  5375 5$:      PUSHR   #^M<R2,R3,R4,R5>       ; Save registers
60      38 A2       0E    28    1B41  5376          MOVC3   #RHDR_C_DATA,CXB$T_R_DATA(R2),(R0) ; Move header info
                    3C    BA    1B46  5377          POPR    #^M<R2,R3,R4,R5>       ; Restore registers
        62    46 A2    9E    1B48  5378 10$:     MOVAB   CXB$T_R_USERDAT(R2),(R2); Set address of received data
  04 A2       3C A3    D0    1B4C  5379          MOVL    IRP$L_XQ_DATBUF(R3),4(R2); Set address of user buffer
              42 A5    A0    1B51  5380          ADDW    UCB$W_DEVBUFSIZ(R5),-  ; Adjust receive buffer quota
           00C8 C5          1B54  5381                  UCB$W_XQ_QUOTA(R5)     ;
        51    1A A2    3C    1B57  5382          MOVZWL  CXB$W_BCNT(R2),R1      ; Find length of received message
                             1B5B  5383 ;
                             1B5B  5384 ; Perform accounting on a per protocol type basis.
                             1B5B  5385 ;
                             1B5B  5386          CNTR    R1,UCB$L_XQ_RBYCTR(R5),L ; Bump the bytes received counter
                             1B67  5387          INCC    UCB$L_XQ_RBLCTR(R5),L  ; Bump the blocks received counter
                             1B71  5388 ;
                             1B71  5389 ; If padding is enabled, then the size of the data is contained in the
                             1B71  5390 ; message as the first word of data.
```

```
                        1B71  5391 ;
                        1B71  5392          ASSUME  NMA$C_STATE_ON EQ 0
                        1B71  5393          ASSUME  NMA$C_STATE_OFF EQ 1
                0F 00D9 C5    E8  1B71  5394          BLBS    UCB$B_XQ_PAD(R5),15$    ; Br if padding is disabled
                50   46 A2    3C  1B76  5395          MOVZWL  CXB$W_R_SIZE(R2),R0     ; Else, pick up real size of
                                  1B7A  5396                                          ;  MSG from the message itself
                   62   02    C0  1B7A  5397          ADDL    #XQ_C_CNTSIZ,(R2)       ; Move pointer past the count field
                                  1B7D  5398 ;
                                  1B7D  5399 ; Verify that the 1st word of data at least makes some sense. The byte
                                  1B7D  5400 ; count for the message must be less than the size of the entire received
                                  1B7D  5401 ; message.
                                  1B7D  5402 ;
                   51   50    B1  1B7D  5403          CMPW    R0,R1                   ; Is size field larger than buffer?
                        31    1E  1B80  5404          BGEQU   45$                     ; Br if yes -
                                  1B82  5405                                          ;  must be strictly Less Than, because
                                  1B82  5406                                          ;  the size field is 2 extra bytes.
                   51   50    3C  1B82  5407          MOVZWL  R0,R1                   ; Else, copy the real buffer size
                                  1B85  5408
                3A A3   01    9B  1B85  5409 15$:     MOVZBW  S^#SS$_NORMAL,IRP$W_XQ_STATUS(R3) ; Assume success
                32 A3   51    B1  1B89  5410          CMPW    R1,IRP$W_BCNT(R3)       ; Request larger than user buffer size?
                        0A    1B  1B8D  5411          BLEQU   20$                     ; Br if no - okay
                     51   32 A3  3C  1B8F  5412          MOVZWL  IRP$W_BCNT(R3),R1       ; Else, set size to minimum of two
             3A A3   0838 8F    B0  1B93  5413          MOVW    #SS$_DATAOVERUN,IRP$W_XQ_STATUS(R3) ; Set return status
                                  1B99  5414 ;
                                  1B99  5415 ; If chained buffers, then setup IRP$W_STS and don't reset the USER BUFFER
                                  1B99  5416 ; SIZE (IRP$W_BCNT), because chained buffers need to have the USER BUFFER
                                  1B99  5417 ; SIZE in IRP$W_BCNT.
                                  1B99  5418 ;
                   10 A2    D5  1B99  5419 20$:     TSTL    CXB$L_LINK(R2)          ; Is this a complex chained buffer?
                        0B    13  1B9C  5420          BEQL    30$                     ; Br if no - set transfer size
                        05    E1  1B9E  5421          BBC     #IRP$V_CHAINED,-        ; Br if user cannot accept complex
                      10 2A A3        1BA0  5422                  IRP$W_STS(R3),45$       ;  chained buffers
                        08    A8  1BA3  5423          BISW    #IRP$M_COMPLX,-         ; Else indicate complex
                   2A A3            1BA5  5424                  IRP$W_STS(R3)           ;    chained buffers
                        04    11  1BA7  5425          BRB     40$                     ; And don't change user buffer size
                32 A3   51    B0  1BA9  5426 30$:     MOVW    R1,IRP$W_BCNT(R3)       ; Set size of transfer
             50   51     10    78  1BAD  5427 40$:     ASHL    #16,R1,R0               ; Set buffer size in status
                        06    12  1BB1  5428          BNEQ    50$                     ; Br if success
                   0054 8F    B0  1BB3  5429 45$:     MOVW    #SS$_CTRLERR,-          ; Set data transfer error
                      3A A3        1BB7  5430                  IRP$W_XQ_STATUS(R3)     ;
                                  1BB9  5431
                50   3A A3    B0  1BB9  5432 50$:     MOVW    IRP$W_XQ_STATUS(R3),R0  ; Get status
                        1D    10  1BBD  5433 60$:     BSBB    IO_DONE                 ; Post the I/O request
                     F87C     31  1BBF  5434          BRW     FILLRCVLIST             ; Fill up the receive buffers
                                  1BC2  5435 ;
                                  1BC2  5436 ; Complete an I/O request packet
                                  1BC2  5437 ;
                                  1BC2  5438 ABORT_PKT:                               ; Abort the I/O request
                   50   2C    9A  1BC2  5439          MOVZBL  S^#SS$_ABORT,R0         ; Return aborted status
                                  1BC5  5440 IO_DONE3:                                ; Complete the I/O request, check for
                                  1BC5  5441                                          ;    timeout
                55   1C A3    D0  1BC5  5442          MOVL    IRP$L_UCB(R3),R5        ; Get UCB address
                51   24 A5    D0  1BC9  5443          MOVL    UCB$L_CRB(R5),R1        ; Get CRB address
                51   10 A1    D0  1BCD  5444          MOVL    CRB$L_AUXSTRUC(R1),R1   ; Get CDB address
             05 0114 C1   09    E1  1BD1  5445          BBC     #XM$V_STS_TIMO,CDB_L_DEVDEPEND(R1),IO_DONE ; Br if not a timeout
                50   022C 8F    3C  1BD7  5446          MOVZWL  #SS$_TIMEOUT,R0         ; Else, return real error code
                                  1BDC  5447 IO_DONE:                                 ; Complete an I/O request
```

```
                38 A3    50    D0  1BDC  5448          MOVL    R0,IRP$L_IOST1(R3)      ; Set status return and size
                                   1BE0  5449 IO_DONE1:                              ; Alternate entry point
                55    1C A3    D0  1BE0  5450          MOVL    IRP$L_UCB(R3),R5        ; Get UCB address
             3C A3    44 A5    D0  1BE4  5451          MOVL    UCB$L_DEVDEPEND(R5),IRP$L_IOST2(R3) ; Set other info
                51    24 A5    D0  1BE9  5452          MOVL    UCB$L_CRB(R5),R1        ; Get CRB address
                51    10 A1    D0  1BED  5453          MOVL    CRB$L_AUXSTRUC(R1),R1   ; Get CDB address
                         06    13  1BF1  5454          BEQL    IO_DONE2                ; Br if no CDB
          3C A3    0114 C1    C8  1BF3  5455          BISL    CDB_L_DEVDEPEND(R1),IRP$L_IOST2(R3) ; Set controller bits
                                   1BF9  5456 IO_DONE2:                              ; P1 IOST2 already setup
                         54    DD  1BF9  5457          PUSHL   R4                      ; Save R4
                54    24 A5    D0  1BFB  5458          MOVL    UCB$L_CRB(R5),R4        ; Get CRB address
                54    10 A4    D0  1BFF  5459          MOVL    CRB$L_AUXSTRUC(R4),R4   ; Get CDB address
                         07    E1  1C03  5460          BBC     #IRP$V_DIAGBUF,-        ; Br if no diagnotic buffer
                   22 2A A3         1C05  5461                  IRP$W_STS(R3),20$       ;
                50    4C A3    D0  1C08  5462          MOVL    IRP$L_DIAGBUF(R3),R0    ; Get diagnostic buffer address
                08 A0    1A    B1  1C0C  5463          CMPW    #RHDR_C_LENGTH,DIAG_W_SIZE(R0) ; Is this a real diag buffer?
                         18    13  1C10  5464          BEQL    20$                     ; Br if not - no diagnostic info
             50    60    08    C1  1C12  5465          ADDL3   #8,(R0),R0              ; Address buffer past start time
          80   00000000'GF    7D  1C16  5466          MOVQ    G^EXE$GQ_SYSTIME,(R0)+  ; Insert stop time
             51    0118 C4    D0  1C1D  5467          MOVL    CDB_L_UCB0(R4),R1       ; Get address of UCB #0
             80    0082 C1    3C  1C22  5468          MOVZWL  UCB$W_ERRCNT(R1),(R0)+  ; Insert error counter
                      0214    30  1C27  5469          BSBW    REG_DUMP                ; Dump registers
                         54 8ED0  1C2A  5470 20$:     POPL    R4                      ; Restore R4
                00000000'GF    17  1C2D  5471          JMP     G^COM$POST              ; Post the I/O and return
```

```
                                        .SBTTL  ASSEM_PKTS - Assemble receive packets
                                ;++
                                ; ASSEM_PKTS - Assemble receive packets
                                ;
                                ; Functional description:
                                ;
                                ; This routine assembles all receive packets into one chain of complex
                                ; buffers.
                                ;
                                ;  Inputs:
                                ;
                                ;       R4 = CDB address
                                ;       R7 = Receive ring address
                                ;
                                ;       IPL = FIPL
                                ;
                                ;  Outputs:
                                ;
                                ;       R0 = Status for request
                                ;       R2 = Address of first receive buffer in chain
                                ;       R6 = Address of last buffer in receive ring
                                ;       R1,R3 are destroyed.
                                ;       All other registers are preserved.
                                ;
                                ;  Implicit Outputs:
                                ;
                                ;       IRP$V_CHAIN and IRP$V_COMPLX bits set in IRP$W_STS if the receive
                                ;       buffer is comprised of complex chained buffers.
                                ;--
                                        .ENABL  LSB
                                ASSEM_PKTS::                                    ; Assemble receive packets
                        0096    30      BSBW    NEXTMSG                         ; Get first message
                     01 50      E8      BLBS    R0,5$                           ; Br if we got one
                                05      RSB                                     ; Return in error
                                ;
                                ; Save number of messages  in chain in CXB$W_R_NCHAIN and total size of
                                ; all messages in CXB$W_BCNT
                                ;
                1C A2    01     B0      MOVW    S^#1,CXB$W_R_NCHAIN(R2) ; Compute total number of buffers
                      1A A2     B4      CLRW    CXB$W_BCNT(R2)          ; Init total size of buffers so far
                50    0C A2     B0      MOVW    CXB$W_LENGTH(R2),R0     ; Get size of message
                         0F     E1      BBC     #RCV_STS_V_LAST,-       ; Br if end of packet
                   51 14 A2             CXB$W_R_STS(R2),40$             ; ..all done with this loop
                         52     DD      PUSHL   R2                      ; Save first receive buffer address
                      53 01     9A      MOVZBL  #MAX_C_CHAIN,R3         ; Allow n messages in chain
                         06 A6  B0      MOVW    RCV_Q_LEN(R6),-         ; Set size of buffer to maximum per rcv
                         0C A2          CXB$Q_LENGTH(R2)                ;
                         52     DD      PUSHL   R2                      ; Save address of current bufr in chain
                          0073  30      BSBW    NEXTMSG                 ; Try for next message
                         51 8ED0 1C59   POPL    R1                      ; Get address of last buffer in chain
                      21 50     E9      BLBC    R0,20$                  ; Toss all messages on error
                   50    6E     D0      MOVL    (SP),R0                 ; Get address of first in chain
                10 A1    52     D0      MOVL    R2,CXB$L_LINK(R1)       ; Store address in chain
                         0C A1  A0      ADDW    CXB$W_LENGTH(R1),-      ; Compute total size of all buffers
                         1A A0          CXB$Q_BCNT(R0)                  ;  in chain - so far
                         1C A0  B6      INCW    CXB$W_R_NCHAIN(R0)      ; Compute number of msgs in chain
                            0F  E1      BBC     #RCV_STS_V_LAST,-       ; Br if end of packet
```

```
                OF 14 A2          1C70  5530                    CXB$W_R_STS(R2),30$     ;
                   D9 53    F5    1C73  5531            SOBGTR  R3,10$                  ; Loop if more than two in chain
                                  1C76  5532 ;
                                  1C76  5533 ; Done with loop and LAST bit still not set - toss all messages
                                  1C76  5534 ;
                                  1C76  5535            INCC    CDB_W_OVRCTR(R4),W      ; Count as hardware error
                                  1C80  5536 ;
                                  1C80  5537 ; Error exit
                                  1C80  5538 ;
                      21    11    1C80  5539 20$:       BRB     TOSAMSG                 ; Toss all messages
                                  1C82  5540
                51    52    D0    1C82  5541 30$:       MOVL    R2,R1                   ; Save address of last message
                      52  8ED0    1C85  5542            POPL    R2                      ; Return first message address in R2
                   0B A1    90    1C88  5543            MOVB    CXB$B_R_FLAGS(R1),-     ; Save only last message buffer info
                   0B A2          1C8B  5544             CXB$B_R_FLAGS(R2)      ;  in first message of chain
                   14 A1    B0    1C8D  5545            MOVW    CXB$W_R_STS(R1),-       ; DITTO
                   14 A2          1C90  5546             CXB$W_R_STS(R2)        ;
             50    0C A1    B0    1C92  5547            MOVW    CXB$W_LENGTH(R1),R0     ; Save size of entire message
                   1A A2    A2    1C96  5548            SUBW    CXB$W_BCNT(R2),-        ; Compute size of last message
                   0C A1          1C99  5549             CXB$W_LENGTH(R1)       ;  and store in CXB format
          1A A2    50    B0    1C9B  5550 40$:       MOVW    R0,CXB$W_BCNT(R2)       ; Return size of complete message
                50    01    9A    1C9F  5551 50$:       MOVZBL  S^#SS$_NORMAL,R0        ; Return success!
                            05    1CA2  5552 100$:      RSB
                                  1CA3  5553 ;
                                  1CA3  5554 ; Toss bad messages
                                  1CA3  5555 ;
                      52  8ED0    1CA3  5556 TOSAMSG:POPL       R2                      ; Restore R2
                                  1CA6  5557 TOSSMSG:INCC       CDB_W_LBECTR(R4),W      ; Up the counter
                   10 A2    DD    1CB0  5558 110$:      PUSHL   CXB$L_LINK(R2)          ; Save address of next in chain
                    F78A    30    1CB3  5559            BSBW    ADDRCVLIST              ; Add buffer to receive list
                      52  8ED0    1CB6  5560            POPL    R2                      ; Restore address of next in chain
                      F5    12    1CB9  5561            BNEQ    110$                    ; Br if more in chain
                      50    D4    1CBB  5562 120$:      CLRL    R0                      ; Assume failure
                      0F    E0    1CBD  5563            BBS     #RCV_STS_V_LAST,-       ; Br if NOT end of chain
                03 08 A6          1CBF  5564             RCV_W_STS(R6),130$     ;  get rest of message
                    FF6E    31    1CC2  5565            BRW     ASSEM_PKTS              ; Else, try for next valid message
                      05    10    1CC5  5566 130$:      BSBB    NEXTMSG                 ; Get next message
                   D8 50    E9    1CC7  5567            BLBC    R0,100$                 ; Br if none
                      EF    11    1CCA  5568            BRB     120$                    ; Check if more possible
                                  1CCC  5569            .DSABL  LSB
                                  1CCC  5570 ;
                                  1CCC  5571 ; Find next message and check ownership
                                  1CCC  5572 ;
                      50    D4    1CCC  5573 NEXTMSG:CLRL       R0                      ; Assume failure
                  010E C4    95    1CCE  5574            TSTB    CDB_B_RCVCNT(R4)        ; Any more receives in progress?
                      01    12    1CD2  5575            BNEQ    5$                      ; Br if yes
                            05    1CD4  5576 1$:        RSB                             ; Else, return
                                  1CD5  5577
                56  010C C4    9A    1CD5  5578 5$:        MOVZBL  CDB_B_LASTRCV(R4),R6    ; Get last ring entry completed inx
                56    7C A446   D0    1CDA  5579            MOVL    CDB_L_RRINGVA(R4)[R6],R6; Get last ring entry address
          05 08 A6    0F    E1    1CDF  5580            BBC     #RCV_STS_V_LAST,RCV_W_STS(R6),10$ ; Br if done
          EB 08 A6    0E    E1    1CE4  5581            BBC     #RCV_STS_V_ERR,RCV_W_STS(R6),1$ ; Br if not done
          0B A6    0A A6    91    1CE9  5582 10$:       CMPB    RCV_W_LENB(R6),RCV_W_LENB+1(R6) ; Are we really done?
                      71    12    1CEE  5583            BNEQ    90$                     ; Br if not, leave now
                52    00FC D4    0F    1CF0  5584            REMQUE  @CDB_Q_RCVPND(R4),R2    ; Get next receive
                      6A    1D    1CF5  5585            BVS     90$                     ; Br if none available (yet)
                   10 A2    D4    1CF7  5586            CLRL    CXB$L_LINK(R2)          ; Assume not a chained buffer
```

```
                     010E C4   97   1CFA  5587          DECB    CDB_B_RCVCNT(R4)          ; One less receive pending
                     010C C4   96   1CFE  5588          INCB    CDB_B_LASTRCV(R4)         ; Bump ring pointer
                       F8 8F   8A   1D02  5589          BICB    #^C<MAX_C_RCV-1>,-        ; Modulo receive ring entry size
                     010C C4         1D05  5590                   CDB_B_LASTRCV(R4)         ;   ...
                                     1D08  5591          CLRBIT  #RCV_DSC_V_VALID,-        ; Indicate that buffer is not valid
                                     1D08  5592                   RCV_W_ADDRHI(R6)          ;
                                     1D0D  5593 ;
                                     1D0D  5594 ; Compute buffer size
                                     1D0D  5595 ;
                     F8FF 8F     AB  1D0D  5596          BICW3   #^C<RCV_STS_M_RLEN>,-     ; Store length <10:8>
                       08 A6         1D11  5597                   RCV_W_STS(R6),-           ;
                       0C A2         1D13  5598                   CXB$W_LENGTH(R2)          ;
                       0A A6   90    1D15  5599          MOVB    RCV_W_LENB(R6),-          ; Store length in CXB <7:0>
                       0C A2         1D18  5600                   CXB$W_LENGTH(R2)          ;
                          2E   A0    1D1A  5601          ADDW    #XQ_C_ADDRCV-<XQ_C_HEADER+XQ_C_CRC>,- ; Add in missed count
                       0C A2         1D1C  5602                   CXB$W_LENGTH(R2)          ;    ..minus header and CRC
                       08 A6   B0    1D1E  5603          MOVW    RCV_W_STS(R6),-           ; Save status flags in CXB
                       14 A2         1D21  5604                   CXB$W_R_STS(R2)
                       01 A6   90    1D23  5605          MOVB    RCV_W_FLAG+1(R6),-        ; Save flags byte (high word)
                       0B A2         1D26  5606                   CXB$B_R_FLAGS(R2)         ;
                                     1D28  5607 ;
                                     1D28  5608 ; Adjust quota and release mapping slot
                                     1D28  5609 ;
                     02F2 C4   95    1D28  5610          TSTB    CDB_B_AQUOTA(R4)          ; Are running on extra QUOTA?
                          06   12    1D2C  5611          BNEQ    30$                       ; Br if not
                     02F2 C4   97    1D2E  5612          DECB    CDB_B_AQUOTA(R4)          ; Else, decrement extra QUOTA
                          07   11    1D32  5613          BRB     40$                       ; Continue
                                     1D34  5614
                     0110 C4   A0    1D34  5615 30$:     ADDW    CDB_W_BSZ(R4),-           ; Replenish CDB quota
                     0112 C4         1D38  5616                   CDB_W_QUOTA(R4)           ;
                   51    22 A2   9A  1D3B  5617 40$:     MOVZBL  CXB$B_XQ_SLOT(R2),R1      ; Get mapping slot number used
                                     1D3F  5618          CLRBIT  R1,CDB_B_RCVMAP(R4)       ; Clear in use flag
                                     1D44  5619          CPUDISP <<790,80$>,-
                                     1D44  5620                  <780,80$>,-
                                     1D44  5621                  <750,80$>,-
                                     1D44  5622                  <730,80$>,-
                                     1D44  5623                  <UV1,100$>>               ; Copy data on u-VAX I
                        50   01   9A  1D5E  5624 80$:     MOVZBL  S^#SS$_NORMAL,R0          ; Return success!
                                05   1D61  5625 90$:     RSB
                                     1D62  5626
                                     1D62  5627 100$:    ;********************
                                     1D62  5628          ; For u-VAX I, ONLY.
                                     1D62  5629          ;********************
                                B3   1D62  5630          BITW    #RCV_STS_M_ESETUP!-       ; Is this a setup packet or error?
                                     1D63  5631                   RCV_STS_M_ERR,-
                   14 A2    6000 8F     1D63  5632                   CXB$W_R_STS(R2)
                                F4   12   1D68  5633          BNEQ    80$                       ; Br if yes, skip it
                    50   00C4 C441   D0   1D6A  5634          MOVL    CDB_L_RCV_VA(R4)[R1],R0   ; Get address of contiguous buffer
                 0110 C4    0C A2   B1   1D70  5635          CMPW    CXB$W_LENGTH(R2),CDB_W_BSZ(R4)  ; Check size of received data
                                07   1B   1D76  5636          BLEQU   110$                      ; Br if okay
                                     1D78  5637          SETBIT  #RCV_STS_V_ERR,-          ; Else, indicate error
                                     1D78  5638                   CXB$W_R_STS(R2)           ;  and let it get tossed
                                DF   11   1D7D  5639          BRB     80$                       ; Continue
                                     1D7F  5640
                                0E   A1   1D7F  5641 110$:    ADDW3   #XQ_C_HEADER,-            ; Add back in the header
                   53    0C A2         1D81  5642                   CXB$W_LENGTH(R2),R3       ;
                                34   BB   1D84  5643          PUSHR   #^M<R2,R4,R5>             ; Save registers
```

```
                38 A2   60    53   28  1D86  5644          MOVC3   R3,(R0),CXB$T_R_DATA(R2) ; Move the data
                        34    BA   1D8B  5645          POPR    #^M<R2,R4,R5>            ; Restore registers
                        CF    11   1D8D  5646          BRB     80$                      ; Continue
```

```
                                1D8F  5648          .SBTTL  MOP_CTR_REQUEST - PROCESS MOP READ COUNTERS REQUEST
                                1D8F  5649 ;++
                                1D8F  5650 ; MOP_CTR_REQUEST - PROCESS MOP READ COUNTERS REQUEST
                                1D8F  5651 ;
                                1D8F  5652 ; FUNCTIONAL DESCRIPTION:
                                1D8F  5653 ;
                                1D8F  5654 ; This routine is called to process a remote request to read the LINE counters.
                                1D8F  5655 ;
                                1D8F  5656 ; Inputs:
                                1D8F  5657 ;
                                1D8F  5658 ;       R2 = Address of the received message
                                1D8F  5659 ;       R4 = CDB address
                                1D8F  5660 ;
                                1D8F  5661 ; Outputs:
                                1D8F  5662 ;
                                1D8F  5663 ;       R0,R1,R2,R3,R5 are destroyed
                                1D8F  5664 ;       R4 is preserved
                                1D8F  5665 ;--
                                1D8F  5666
                                1D8F  5667 MOP_CTR_REQUEST::                    ; Process a read counters request
                      10 A2  D5 1D8F  5668          TSTL    CXB$L_LINK(R2)    ; Is this a chained message?
                         24  12 1D92  5669          BNEQ    80$               ; Br if yes, return packets
                             B1 1D94  5670          CMPW    #IRP$C_LENGTH-    ; Is a receive buffer large enough?
                                1D95  5671                  +CXB$C_HEADER+3-
                                1D95  5672                  +MOP_CTR_SIZE+8,-
                   08 A2  0168  8F  1D95  5673                  CXB$W_SIZE(R2)
                             1C  1A 1D9A  5674          BGTRU   80$               ; Br if no, ignore message
                      55  0118  C4 1D9C  5675          MOVL    CDB_L_UCB0(R4),R5 ; Get address of UCB #0
                        50   3E  A2 1DA1  5676          MOVQ    CXB$G_R_SRC(R2),R0 ; Save source node address
                             47  A2 1DA5  5677          MOVW    CXB$T_R_USERDAT+1(R2),-; Copy the request ID
                            010D C2 1DA8  5678                  IRP$C_LENGTH+CXB$C_HEADER+1(R2) ;  to the Message block
                             05D2  30 1DAB  5679          BSBW    BLD_IRP           ; Turn the message block into an IRP
                      40 A3   50  7D 1DAE  5680          MOVQ    R0,IRP$Q_STATION(R3) ; Set the return node address
                      14 A3   54  D0 1DB2  5681          MOVL    R4,IRP$L_ASTPRM(R3) ; Save the CDB address
                             0C  11 1DB6  5682          BRB     MOP_CTR_BUILD     ; Build the response message
                                1DB8  5683 ;
                                1DB8  5684 ; Error - returns packets to receive queue
                                1DB8  5685 ;
                      10 A2     DD 1DB8  5686 80$:       PUSHL   CXB$L_LINK(R2)    ; Save next in chain
                       F682     30 1DBB  5687          BSBW    ADDRCVLIST        ; Add buffer to receive list
                         52 8ED0  1DBE  5688          POPL    R2                ; Restore next buffer
                         F5  12 1DC1  5689          BNEQ    80$               ; Loop if more
                             05 1DC3  5690          RSB                       ; Return to caller
                                1DC4  5691
```

```
                     1DC4  5693          .SBTTL  MOP_CTR_BUILD - BUILD THE MOP COUNTER RETURN MESSAGE
                     1DC4  5694 ;++
                     1DC4  5695 ; MOP_CTR_BUILD - BUILD THE MOP COUNTER RETURN MESSAGE
                     1DC4  5696 ;
                     1DC4  5697 ; FUNCTIONAL DESCRIPTION:
                     1DC4  5698 ;
                     1DC4  5699 ; This routine is called to build the return message in response to a
                     1DC4  5700 ; MOP read counters request.
                     1DC4  5701 ;
                     1DC4  5702 ; Inputs:
                     1DC4  5703 ;
                     1DC4  5704 ;       R3 = IRP address
                     1DC4  5705 ;       R4 = CDB address
                     1DC4  5706 ;       R5 = UCB address of UNIT 0
                     1DC4  5707 ;
                     1DC4  5708 ; Outputs:
                     1DC4  5709 ;
                     1DC4  5710 ;       R4 = CDB address
                     1DC4  5711 ;       R5 = UCB address of UNIT 0
                     1DC4  5712 ;
                     1DC4  5713 ;       R0,R1 are destroyed
                     1DC4  5714 ;--
                     1DC4  5715
                     1DC4  5716 MOP_CTR_BUILD:                          ; Build the MOP counter return msg
                     1DC4  5717
     01F8 8F    BB   1DC4  5718          PUSHR   #^M<R3,R4,R5,R6,R7,R8>  ; Save registers
     56   54    D0   1DC8  5719          MOVL    R4,R6                   ; Copy CDB address
 53  010C C3    9E   1DCB  5720          MOVAB   IRP$C_LENGTH+-          ; Point to start of message
                     1DD0  5721                  CXB$C_HEADER(R3),R3     ;    block
          83    0B   90   1DD0  5722          MOVB    #NI_CTR_REPLY,(R3)+     ; Set function to reply
          83    B5   1DD3  5723          TSTW    (R3)+                   ; Skip request ID (filled in earlier)
     57  E398 CF    9E   1DD5  5724          MOVAB   MOPCTRTAB,R7            ; Get address of MOP counters
          58    87   3C   1DDA  5725 20$:     MOVZWL  (R7)+,R8                ; Get the offset to the counter desired
          19    13   1DDD  5726          BEQL    30$                     ; Br if end of table
          58    56   C0   1DDF  5727          ADDL    R6,R8                   ; Compute address of the counter
 52   67    07    00   EF   1DE2  5728          EXTZV   #0,#7,(R7),R2           ; Get the width field without bitmap
      63    68    52   28   1DE7  5729          MOVC3   R2,(R8),(R3)            ; Copy the counter
          51    87   9A   1DEB  5730          MOVZBL  (R7)+,R1                ; Get the width field again
        E8  51    07   E1   1DEE  5731          BBC     #7,R1,20$               ; Br if no BITMAP to return
        83    FE  A8   B0   1DF2  5732          MOVW    -2(R8),(R3)+            ; Else, store the BITMAP
              E2    11   1DF6  5733          BRB     20$                     ; Loop for next counter
                     1DF8  5734
          01F8 8F    BA   1DF8  5735 30$:     POPR    #^M<R3,R4,R5,R6,R7,R8>  ; Restore registers
                     1DFC  5736
 0C A3    2339'CF    9E   1DFC  5737          MOVAB   W^DELETE_BLOCK,IRP$L_PID(R3) ; Set the return address
 52       00C4 C3    9E   1E02  5738          MOVAB   IRP$C_LENGTH(R3),R2     ; Get address of CXB
    0A A2    1B    90   1E07  5739          MOVB    #DYN$C_CXB,CXB$B_TYPE(R2) ; Make it look like a CXB
          005C 8F    B0   1E0B  5740          MOVW    #MOP_CTR_SIZE+8+3,-     ; Set size of transfer
            1A A2         1E0F  5741                  CXB$Q_BCNT(R2)          ;
                     1E11  5742          ASSUME  CXB$C_HEADER  EQ  CXB$T_DATA+XQ_C_HEADER
              3A   B0   1E11  5743          MOVW    #CXB$T_DATA,-           ; Set offset to start of data
            18 A2         1E13  5744                  CXB$W_BOFF(R2)
        24 A2    53   D0   1E15  5745          MOVL    R3,CXB$L_IRP(R2)        ; Save IRP address in CXB
           40 A3    7D   1E19  5746          MOVQ    IRP$Q_STATION(R3),-     ; Set STATION in CXB
           28 A2         1E1C  5747                  CXB$Q_STATION(R2)
              01    90   1E1E  5748          MOVB    #XQ_FC_V_XMIT,-         ; Set function request in CXB
            20 A2         1E20  5749                  CXB$B_XQ_FUNC(R2)
```

```
                55   1C A3    D0  1E22  5750          MOVL    IRP$L_UCB(R3),R5        ; Get the UCB address
                52   3A A2    9E  1E26  5751          MOVAB   CXB$T_T_DATA(R2),R2     ; Set R2 to start of data
                     40 A3    7D  1E2A  5752          MOVQ    IRP$Q_STATION(R3),-     ; Set th destination address
                        62           1E2D  5753                  XBUF_G_DEST(R2)
                   0260 8F    B0  1E2E  5754          MOVW    #NI_CTR_PROTYP,-        ; Store the protocol type
                     0C A2          1E32  5755                  XBUF_W_TYPE(R2)
                   00C4 C3    0E  1E34  5756          INSQUE  IRP$C_LENGTH(R3),-      ; Insert request on request queue
                   00E0 D4          1E38  5757                  @CDB_Q_XMTREQ+4(R4)
                      E655    31  1E3B  5758          BRW     XMT_ALT_START           ; Startup the reply
                              1E3E  5759
```

```
                                1E3E  5761          .SBTTL  REG_DUMP - DEQNA ERROR LOG AND DIAGNOSTICS REGISTER DUMP
                                1E3E  5762 ;++
                                1E3E  5763 ; REG_DUMP - DIAGNOSTICS REGISTER DUMP ROUTINE
                                1E3E  5764 ;
                                1E3E  5765 ; FUNCTIONAL DESCRIPTION:
                                1E3E  5766 ;
                                1E3E  5767 ; This routine is used to return the DEQNA error log and diagnostics
                                1E3E  5768 ; buffer on error or diagnostic QIO function.
                                1E3E  5769 ;
                                1E3E  5770 ; Inputs:
                                1E3E  5771 ;
                                1E3E  5772 ;       R0 = Address of the buffer @ DIAG_L_EXTRA
                                1E3E  5773 ;       R4 = CDB address
                                1E3E  5774 ;       R5 = UCB address of the unit
                                1E3E  5775 ;
                                1E3E  5776 ; Outputs:
                                1E3E  5777 ;
                                1E3E  5778 ;       R0,R1 are destroyed
                                1E3E  5779 ;       R4,R5 are preserved
                                1E3E  5780 ;--
                                1E3E  5781 REG_DUMP::
                                1E3E  5782          ASSUME  DIAG_L_DEPEND EQ DIAG_L_EXTRA+4
                                1E3E  5783          ASSUME  DIAG_W_CSR EQ DIAG_L_DEPEND
                                1E3E  5784          ASSUME  DIAG_W_ERR EQ DIAG_W_CSR+2
                                1E3E  5785          ASSUME  DIAG_W_ERR2 EQ DIAG_W_ERR+2
                                1E3E  5786          ASSUME  DIAG_G_HWA EQ DIAG_W_ERR2+2
    80    06    9A  1E3E  5787          MOVZBL  #DIAG_C_EXTRA,(R0)+     ; Insert number of returned long words
  80    10 A4    B0  1E41  5788          MOVW    CDB_L_CSR(R4),(R0)+     ; Insert last CSR contents
80    011D C4    9B  1E45  5789          MOVZBW  CDB_B_DIAG1(R4),(R0)+   ; Return error flags info
80    011E C4    B0  1E4A  5790          MOVW    CDB_W_DIAG2(R4),(R0)+   ; Return extra error info
80    0254 C4    D0  1E4F  5791          MOVL    CDB_G_HWA(R4),(R0)+     ; Return hardware physical address
80    0258 C4    B0  1E54  5792          MOVW    CDB_G_HWA+4(R4),(R0)+   ;       ...
                   05  1E59  5793          RSB
```

```
                                1E5A  5795          .SBTTL  RESTART_ROUT - PROCESS EXPIRATION OF RESTART TIMER
                                1E5A  5796 ;++
                                1E5A  5797 ; RESTART_ROUT - PROCESS EXPIRATION OF RESTART TIMER
                                1E5A  5798 ;
                                1E5A  5799 ; Functional description:
                                1E5A  5800 ;
                                1E5A  5801 ; This routine is entered when the RESTART delta time has expired. The action
                                1E5A  5802 ; is to check if the specified unit has been restarted, if so, then it is
                                1E5A  5803 ; automatically restarted.
                                1E5A  5804 ;
                                1E5A  5805 ; Inputs:
                                1E5A  5806 ;
                                1E5A  5807 ;       R4 = CDB address
                                1E5A  5808 ;       R5 = TQE address (but must be at least as long as an IRP)
                                1E5A  5809 ;
                                1E5A  5810 ;       IPL = IPL$_TIMER
                                1E5A  5811 ;
                                1E5A  5812 ; Implicit inputs:
                                1E5A  5813 ;
                                1E5A  5814 ;       IRP$L_RBOFF(R5) = UCB address for UNIT
                                1E5A  5815 ;
                                1E5A  5816 ; Outputs:
                                1E5A  5817 ;
                                1E5A  5818 ;       R0-R3 are destroyed.
                                1E5A  5819 ;--
                                1E5A  5820          .ENABL  LSB
                                1E5A  5821 RESTART_ROUT::                           ; Process expiration of restart timer
                     55   DD    1E5A  5822          PUSHL   R5                      ; Save R5
                                1E5C  5823          ASSUME  IRP$L_RBOFF GT TQE$C_LENGTH
                52   30 A5   9E 1E5C  5824          MOVAB   TQE$C_LENGTH(R5),R2     ; Point to IRP portion of TQE
             55   0098 C2   D0 1E60  5825          MOVL    IRP$L_RBOFF(R2),R5      ; Reset R5 to UCB address
                                1E65  5826 ;
                                1E65  5827 ; Turn TQE into an IRP
                                1E65  5828 ;
                     53   82   7E 1E65  5829          MOVAQ   (R2)+,R3                ; Copy IRP address, skip to size field
                                1E68  5830          ASSUME  IRP$W_SIZE EQ 8
                                1E68  5831          ASSUME  IRP$B_TYPE EQ IRP$W_SIZE+2
                                1E68  5832          ASSUME  IRP$B_RMOD EQ IRP$B_TYPE+1
                     52   02   C0 1E68  5833          ADDL    #2,R2                   ; Skip size field
                     82   0A   B0 1E6B  5834          MOVW    #DYN$C_IRP,(R2)+        ; Set type to IRP
                                1E6E  5835          ASSUME  IRP$L_PID EQ IRP$B_RMOD+1
          82   1EB3'CF   9E 1E6E  5836          MOVAB   W^RETURN_IRP,(R2)+      ; Set return address form IOPOST
                                1E73  5837          ASSUME  IRP$L_AST EQ IRP$L_PID+4
                                1E73  5838          ASSUME  IRP$L_ASTPRM EQ IRP$L_AST+4
                     82   7C   1E73  5839          CLRQ    (R2)+                   ; Clear AST, ASTPRM
                                1E75  5840          ASSUME  IRP$L_WIND EQ IRP$L_ASTPRM+4
                                1E75  5841          ASSUME  IRP$L_UCB EQ IRP$L_WIND+4
                     82   D4   1E75  5842          CLRL    (R2)+                   ; Clear WIND
                     82   55   D0 1E77  5843          MOVL    R5,(R2)+                ; Set UCB address
                                1E7A  5844          ASSUME  IRP$W_FUNC EQ IRP$L_UCB+4
                                1E7A  5845          ASSUME  IRP$B_XQ_FUNC EQ IRP$W_FUNC+1
                                1E7A  5846          ASSUME  IRP$B_EFN EQ IRP$W_FUNC+2
                                1E7A  5847          ASSUME  IRP$B_PRI EQ IRP$B_EFN+1
                                1E7A  5848          ASSUME  IRP$L_IOSB EQ IRP$B_PRI+1
          82   0500 8F   B0 1E7A  5849          MOVW    #<XQ_FC_V_RESTART@8>,(R2)+ ; Set function request
                     82   B4   1E7F  5850          CLRW    (R2)+                   ; Clear EFN, PRI
                     82   D4   1E81  5851          CLRL    (R2)+                   ; Clear IOSB
```

```
                        1E83  5852          ASSUME  IRP$W_CHAN EQ IRP$L_IOSB+4
                        1E83  5853          ASSUME  IRP$W_STS EQ IRP$W_CHAN+2
                        1E83  5854          ASSUME  IRP$L_SVAPTE EQ IRP$W_STS+2
                82   7C 1E83  5855          CLRQ    (R2)+                   ; Clear CHAN, STS, SVAPTE
                        1E85  5856          ASSUME  IRP$W_BOFF EQ IRP$L_SVAPTE+4
                        1E85  5857          ASSUME  IRP$W_BCNT EQ IRP$W_BOFF+2
                        1E85  5858          ASSUME  IRP$W_BCNT EQ IRP$L_BCNT
                        1E85  5859          ASSUME  IRP$L_MEDIA EQ IRP$W_BCNT+6
                82   7C 1E85  5860          CLRQ    (R2)+                   ; Clear BOFF, BCNT
             82 00   3C 1E87  5861          MOVZWL  #<XQ_FC_V_INIT@8>,(R2)+ ; Set MEDIA
                82   D4 1E8A  5862          CLRL    (R2)+                   ; Clear MEDIA+4
                        1E8C  5863 ;
                        1E8C  5864 ; RESTART the UNIT
                        1E8C  5865 ;
                        1E8C  5866          DSBINT  UCB$B_FIPL(R5)          ; Raise IPL
           4000 8F   AA 1E93  5867          BICW    #UCB$M_XQ_INTERLOCK,-   ; Clear the RESTART interlock
              68 A5       1E97  5868                  UCB$W_DEVSTS(R5)        ;
                   00   E2 1E99  5869          BBSS    #UCB$V_XQ_INITED,-      ; Br if unit already inited
           09 68 A5       1E9B  5870                  UCB$W_DEVSTS(R5),10$    ;
                 F02D   30 1E9E  5871          BSBW    START                   ; Start protocol
                08 50   E8 1EA1  5872          BLBS    R0,30$                  ; Br if success
                        1EA4  5873
                 F00D   30 1EA4  5874          BSBW    STOP                    ; Shutdown unit
             55    53   D0 1EA7  5875 10$:     MOVL    R3,R5                   ; Point R5 to IRP
                   07   10 1EAA  5876          BSBB    RETURN_IRP              ; Return the IRP
                        1EAC  5877 30$:     ENBINT                          ; Re-enable interrupts
                55 8ED0 1EAF  5878          POPL    R5                      ; Restore R5
                      05 1EB2  5879          RSB                             ; Return to caller
                        1EB3  5880
                        1EB3  5881 RETURN_IRP:
           50    D0 A5   9E 1EB3  5882          MOVAB   -TQE$C_LENGTH(R5),R0    ; Get address of start of structure
00000000'GF      17   1EB7  5883          JMP     G^COM$DRVDEALMEM        ; Deallocate the IRP
```

```
                     1EBD  5885          .SBTTL  TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
                     1EBD  5886 ;++
                     1EBD  5887 ; TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
                     1EBD  5888 ;
                     1EBD  5889 ; Functional description:
                     1EBD  5890 ;
                     1EBD  5891 ; This routine is entered when the TQE delta time has expired. The action is to
                     1EBD  5892 ; check all timer cells and shut down the controller if any have expired.
                     1EBD  5893 ;
                     1EBD  5894 ; Inputs:
                     1EBD  5895 ;
                     1EBD  5896 ;       R4 = CDB address
                     1EBD  5897 ;       R5 = TQE address
                     1EBD  5898 ;
                     1EBD  5899 ;       IPL = IPL$_TIMER
                     1EBD  5900 ;
                     1EBD  5901 ; Outputs:
                     1EBD  5902 ;
                     1EBD  5903 ;       R0-R3 are destroyed.
                     1EBD  5904 ;       R4,R5 are preserved
                     1EBD  5905 ;--
                     1EBD  5906          .ENABL  LSB
                     1EBD  5907 TQE_TIMER::                                  ; Process expiration of TQE timer
 50   0118 C4   D0   1EBD  5908          MOVL    CDB_L_UCB0(R4),R0           ; Get first UCB address
                     1EC2  5909          DSBINT  UCB$B_DIPL(R0)              ; Sync access to UCB and CDB
           00   E1   1EC9  5910          BBC     #CDB_STS_V_INITED,-         ; Br if NOT inited
     65 024A C4       1ECB  5911                  CDB_B_STS(R4),STOP_TQE
      020E C4   95   1ECF  5912          TSTB    CDB_B_TIM_XMT(R4)           ; Is the xmit timer going?
           06   13   1ED3  5913          BEQL    10$                         ; Br if not
      020E C4   97   1ED5  5914          DECB    CDB_B_TIM_XMT(R4)           ; Timer expired?
           4E   13   1ED9  5915          BEQL    TIMOUT                      ; Br if yes
           0C   E1   1EDB  5916 10$:     BBC     #XM$V_STS_BUFFAIL,-         ; Br if NO buffer allocation failures
     07 0114 C4       1EDD  5917                  CDB_L_DEVDEPEND(R4),30$ ;
                     1EE1  5918          SETIPL  CDB_B_FIPL(R4)              ; Else, sync access to UCB & CDB
          F556   30   1EE5  5919          BSBW    FILRCVLIST                  ; And try to replenish receive buffers
                     1EE8  5920 30$:     ENBINT                              ; Restore IPL
                     1EEB  5921
                     1EEB  5922          .IF DF  POINT
                     1EEB  5923          PUSHQ   R4                          ;% Save R4, R5
 55   0118 C4   D0   1EEE  5924          MOVL    CDB_L_UCB0(R4),R5           ;% Get UCB address of unit 0
                     1EF3  5925          DSBINT  UCB$B_FIPL(R5)              ;% Sync access to UCB
   55   30 A5   D0   1EFA  5926 50$:     MOVL    UCB$L_LINK(R5),R5           ;% Travel down UCBs
           22   13   1EFE  5927          BEQL    70$                         ;% Br if end of list
                     1F00  5928          ASSUME  UCB$V_XQ_INITED EQ 0
      F6 68 A5   E9   1F00  5929          BLBC    UCB$W_DEVSTS(R5),50$        ;% Br if not inited
           04   E1   1F04  5930          BBC     #UCB$V_XQ_RUN,-             ;% Br if not running yet
      F1 68 A5         1F06  5931                  UCB$W_DEVSTS(R5),50$        ;%
           00   91   1F09  5932          CMPB    #NMA$C_LINPR_POI,-          ;% Are we in point to point mode?
      00D8 C5         1F0B  5933                  UCB$B_XQ_PRO(R5)            ;%
           EA   12   1F0E  5934          BNEQ    50$                         ;% Br if not
                B3   1F10  5935          BITW    #UCB$M_XQ_START!-           ;% Br if not in startup or
                     1F11  5936                    UCB$M_XQ_STACK,-          ;%  stack wait states
 68 A5    0060 8F         1F11  5937                  UCB$W_DEVSTS(R5)        ;%
           E2   13   1F16  5938          BEQL    50$                         ;%
   52   0191 C5   D0   1F18  5939          MOVL    UCB$L_XQ_STIRP(R5),R2       ;% Get startup IRP addresss
          0422   30   1F1D  5940          BSBW    BLD_STRT_IRP                ;% Build the startup IRP
           D8   11   1F20  5941          BRB     50$                         ;% Look for more
```

```
                     1F22  5942 70$:    ENBINT                          ;% Re-enable interrupts
                     1F25  5943         POPQ    R4                      ;% Restore R4, R5
                     1F28  5944         .ENDC
                     1F28  5945
                  05 1F28  5946         RSB                             ; Return to caller
                     1F29  5947
                     1F29  5948 TIMOUT: PUSHQ   R4                      ; Save R4, R5
                  55 50 D0  1F2C  5949         MOVL    R0,R5             ; Copy address of UCB0
                     26 10  1F2F  5950         BSBB    DEV_TIMEOUT       ; Else, timeout has occurred
                     1F31  5951         POPQ    R4                      ; Restore R4, R5
                     1F34  5952
                     1F34  5953 STOP_TQE:                               ; Stop the TQE
               0B A5 04 8A 1F34  5954         BICB    #TQE$M_REPEAT,TQE$B_RQTYPE(R5) ; Stop the timer
                     08 8A 1F38  5955         BICB    #CDB_STS_M_TIMER,-      ; Indicate that timer is stopped
                  024A C4  1F3A  5956                 CDB_B_STS(R4)           ;
                     04 8A 1F3D  5957         BICB    #DPT$M_NOUNLOAD,-       ; OKAY to unload the driver now
          0000000D'EF     1F3F  5958                 DPT$TAB+DPT$B_FLAGS     ;
                     A2 11 1F44  5959         BRB     30$                     ; Leave
                     1F46  5960         .DSABL  LSB
```

```
                                1F46  5962          .SBTTL  TIMEOUT - TIMEOUT SERVICE ROUTINE
                                1F46  5963 ;++
                                1F46  5964 ; TIMEOUT - TIMEOUT SERVICE ROUTINE
                                1F46  5965 ;
                                1F46  5966 ; Functional description:
                                1F46  5967 ;
                                1F46  5968 ; This routine is entered on device timeout. The action is to
                                1F46  5969 ; shut the unit down.
                                1F46  5970 ;
                                1F46  5971 ; Inputs:
                                1F46  5972 ;
                                1F46  5973 ;       R5 = UCB ADDRESS
                                1F46  5974 ;
                                1F46  5975 ;       IPL = DIPL
                                1F46  5976 ;
                                1F46  5977 ; Outputs:
                                1F46  5978 ;
                                1F46  5979 ;       R3,R4 are destroyed.
                                1F46  5980 ;       R5 is preserved
                                1F46  5981 ;--
                                1F46  5982          .ENABL  LSB
                                1F46  5983 TIMEOUT::                                ; Timeout or powerfail
    64 A5    03    AA           1F46  5984          BICW    #UCB$M_TIM!UCB$M_INT,UCB$W_STS(R5) ; Disable timer
                                1F4A  5985
                                1F4A  5986          ASSUME  UCB$V_XQ_INITED  EQ  0
      32 68 A5    E9            1F4A  5987          BLBC    UCB$W_DEVSTS(R5),20$    ; Br if not inited
 53     01    11    78           1F4E  5988          ASHL    #XQ_SOFT_V_POWER+16,#1,R3 ; Assume powerfail
1C 64 A5    05    E0            1F52  5989          BBS     #UCB$V_POWER,UCB$W_STS(R5),10$ ; Br if powerfail
                                1F57  5990 DEV_TIMEOUT::                            ; Hardware device timeout
    54    24 A5    D0           1F57  5991          MOVL    UCB$L_CRB(R5),R4        ; Get CRB address
                                1F5B  5992          ASSUME  IDB$L_CSR EQ 0
    53    2C B4    D0           1F5B  5993          MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R3 ; Get CSR address
    0E A3    02    B0           1F5F  5994          MOVW    #XQ_CSR_M_RESET,CSR(R3) ; Stop the device
    54    10 A4    D0           1F63  5995          MOVL    CRB$L_AUXSTRUC(R4),R4   ; Get CDB address
                17    13        1F67  5996          BEQL    20$                     ; Br if no CDB
 53     01    10    78           1F69  5997          ASHL    #XQ_SOFT_V_TIMEOUT+16,#1,R3 ; Indicate timeout
                               1F6D  5998          SETBIT  #XM$V_STS_TIMO,CDB_L_DEVDEPEND(R4) ; Set error status
 53   4000 8F    B0           1F73  5999 10$:     MOVW    #XQ_CSR_M_ERR,R3        ; Indicate fatal error
           55    DD           1F78  6000          PUSHL   R5                      ; Save UCB address
        F746    30           1F7A  6001          BSBW    SCHED_FORK              ; Schedule a fork process
           55 8ED0           1F7D  6002          POPL    R5                      ; Restore UCB address
                05           1F80  6003 20$:     RSB                             ; Return to caller
                             1F81  6004          .DSABL  LSB
```

```
                                1F81  6006          .SBTTL  ALLOC_CDB - ALLOCATE THE CDB
                                1F81  6007 ;++
                                1F81  6008 ; ALLOC_CDB - ALLOCATE THE CDB
                                1F81  6009 ;
                                1F81  6010 ; Functional description:
                                1F81  6011 ;
                                1F81  6012 ; This routine allocates and initializes the CDB.
                                1F81  6013 ;
                                1F81  6014 ; Inputs:
                                1F81  6015 ;
                                1F81  6016 ;       R5 = UCB address
                                1F81  6017 ;
                                1F81  6018 ; Outputs:
                                1F81  6019 ;
                                1F81  6020 ;       R0 = Status return for request
                                1F81  6021 ;
                                1F81  6022 ;       All other registers are preserved.
                                1F81  6023 ;
                                1F81  6024 ;--
                                1F81  6025
                                1F81  6026 ALLOC_CDB:                                   ; Allocate a CDB
             3E     BB          1F81  6027          PUSHR   #^M<R1,R2,R3,R4,R5>         ; Save registers
     51   02F4   8F     3C      1F83  6028          MOVZWL  #CDB_C_LENGTH,R1            ; Get size of CDB allocation
      00000000'GF     16      1F88  6029          JSB     G^EXE$ALONONPAGED           ; Try to allocate CDB
            42     50   E9      1F8E  6030          BLBC    R0,90$                      ; Br if error
                                1F91  6031
       54    24     A5   D0     1F91  6032          MOVL    UCB$L_CRB(R5),R4            ; Get CRB address
       10   A4      52   D0     1F95  6033          MOVL    R2,CRB$L_AUXSTRUC(R4)       ; Store CDB address
       54    28     A5   D0     1F99  6034          MOVL    UCB$L_DDB(R5),R4            ; Get DDB address
       54    04     A4   D0     1F9D  6035          MOVL    DDB$L_UCB(R4),R4            ; Get UCB0 address
          53    54   D0     1FA1  6036          MOVL    R4,R3                       ; Save UCB0 address
             0254   C2     9E  1FA4  6037 10$:     MOVAB   CDB_G_HWA(R2),-             ; Store address of NI device's
             0090   C4         1FA8  6038                  UCB$L_NI_HWAPTR(R4)         ; unique hardware address
          54    30     A4   D0     1FAB  6039          MOVL    UCB$L_LINK(R4),R4           ; Position to next UCB
                F3     12     1FAF  6040          BNEQ    10$                         ; Continue if more UCBs
                                1FB1  6041
                                1FB1  6042 ;
                                1FB1  6043 ; Initialize CDB
                                1FB1  6044 ;
                                1FB1  6045 ;   The PADDING MODE and the ECHO MODE of the QNA will default to
                                1FB1  6046 ;   the enabled (ON) state.
                                1FB1  6047 ;
                                1FB1  6048          ASSUME  NMA$C_STATE_ON EQ 0
                                1FB1  6049
                                1FB1  6050          PUSHQ   R2                          ; Save CDB, UCB0 address
  62   51   00   62   00   2C  1FB4  6051          MOVC5   #0,(R2),#0,R1,(R2)          ; Zero the structure
                                1FBA  6052          POPQ    R2                          ; Restore CDB, UCB0 address
                                1FBD  6053
             0118   C2   53   D0  1FBD  6054          MOVL    R3,CDB_L_UCB0(R2)           ; Save address of UCB0
                                1FC2  6055 ;
                                1FC2  6056 ;  Init CDB fork block.
                                1FC2  6057 ;
                                1FC2  6058          ASSUME  CDB_L_FQFL EQ 0
                                1FC2  6059          ASSUME  CDB_L_FQBL EQ CDB_L_FQFL+4
                     82   7C    1FC2  6060          CLRQ    (R2)+                       ; Skip link pointers
                                1FC4  6061          ASSUME  CDB_W_SIZE EQ CDB_L_FQBL+4
                                1FC4  6062          ASSUME  CDB_B_TYPE EQ CDB_W_SIZE+2
```

```
                                1FC4  6063          ASSUME  CDB_B_FIPL EQ CDB_B_TYPE+1
            82    083302F4 8F   D0  1FC4  6064          MOVL    #<<^IPL$_XQ_FIPL@8>!DYN$C_CDB>@16>!- ; Set structure type and FIPL
                                1FCB  6065                  CDB_C_LENGTH,(R2)+          ;  and size
                                1FCB  6066          ASSUME  CDB_L_FPC EQ CDB_B_FIPL+1
                82    F711 CF   9E  1FCB  6067          MOVAB   FORK_PROC,(R2)+             ; Set fork process address
                      50   01   9A  1FD0  6068          MOVZBL  S^#SS$_NORMAL,R0            ; Return success
                           3E   BA  1FD3  6069 90$:     POPR    #^M<R1,R2,R3,R4,R5>         ; Restore registers
                                05  1FD5  6070          RSB                                 ; Return to caller
```

```
                     1FD6  6072          .SBTTL  SHUTDOWN_QNA - SHUTDOWN QNA AND ALL UNITS
                     1FD6  6073 ;++
                     1FD6  6074 ; SHUTDOWN_QNA - SHUTDOWN QNA AND ALL UNITS
                     1FD6  6075 ;
                     1FD6  6076 ; Inputs:
                     1FD6  6077 ;
                     1FD6  6078 ;       R4 = CDB address
                     1FD6  6079 ;       R5 = UCB address of unit #0
                     1FD6  6080 ;
                     1FD6  6081 ;
                     1FD6  6082 ;       IPL = FIPL
                     1FD6  6083 ;
                     1FD6  6084 ; Outputs:
                     1FD6  6085 ;
                     1FD6  6086 ;       R3-R5 are preserved.
                     1FD6  6087 ;
                     1FD6  6088 ;       R0-R2 are destroyed.
                     1FD6  6089 ;--
                     1FD6  6090
                     1FD6  6091 SHUTDOWN_QNA::                          ; Shutdown QNA
          00   E0    1FD6  6092          BBS     #CDB_STS_V_INITED,-     ; Br if QNA inited
01 024A C4           1FD8  6093                  CDB_B_STS(R4),10$       ;
               05    1FDC  6094          RSB                            ; Else, return
                     1FDD  6095
     00E8 8F   BB    1FDD  6096 10$:     PUSHR   #^M<R3,R5,R6,R7>       ; Save registers
                     1FE1  6097 ;
                     1FE1  6098 ; Shutdown QNA and reset controller status
                     1FE1  6099 ;
     52   24 A5  D0  1FE1  6100          MOVL    UCB$L_CRB(R5),R2       ; Get CRB address
                     1FE5  6101          ASSUME  IDB$L_CSR EQ 0
     52   2C B2  D0  1FE5  6102          MOVL    @CRB$L_INTD+VEC$L_IDB(R2),R2 ; Get CSR address
                     1FE9  6103          DSBINT  UCB$B_DIPL(R5)         ; Raise IPL for master clear
     0E A2    02  A8  1FF0  6104          BISW    #XQ_CSR_M_RESET,CSR(R2) ; Disable device
024A C4   F3 8F  8A  1FF4  6105          BICB    #^C<CDB_STS_M_FORK_PEND!- ; Reset all but needed bits
                     1FFA  6106                  CDB_STS_M_TIMER>,CDB_B_STS(R4) ;
                     1FFA  6107          ENBINT                         ; Return to fork level
                     1FFD  6108 ;
                     1FFD  6109 ; Release the receive and transmit buffer map registers
                     1FFD  6110 ;
          57   D4    1FFD  6111          CLRL    R7                     ; Init slot number
                     1FFF  6112          ASSUME  CDB_L_RCVMAP+<4*<MAX_C_RCV-1>> EQ CDB_L_XMTMAP
     56   1C A4  9E  1FFF  6113          MOVAB   CDB_L_RCVMAP(R4),R6    ; Get address of mappig slots
     53   24 A5  D0  2003  6114 20$:     MOVL    UCB$L_CRB(R5),R3       ; Get CRB address
                     2007  6115          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                     2007  6116          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
     34 A3    86  D0  2007  6117          MOVL    (R6)+,CRB$L_INTD+VEC$W_MAPREG(R3) ; Set mapping information
              1B  19  200B  6118          BLSS    30$                    ; Br if none allocated
          37 A3  95  200D  6119          TSTB    CRB$L_INTD+VEC$B_DATAPATH(R3) ; Is there a datapath?
              0C  13  2010  6120          BEQL    25$                    ; Br if not - don't do purge or release
                     2012  6121          PURDPR                         ; Purge the data path
                     2018  6122          RELDPR                         ; Release the data path
                     201E  6123 25$:     RELMPR                         ; Release the map register
     FC A6    01  CE  2024  6124          MNEGL   #1,-4(R6)              ; Reset mapping info
                     2028  6125          ASSUME  MAX_C_RCV EQ 8
                     2028  6126 30$:     CLRBIT  R7,CDB_B_RCVMAP(R4)    ; Clear mapping slot flag
     D2 57    0C  F2  202D  6127          AOBLSS  #MAX_C_RCV+MAX_C_XMT,R7,20$ ; Loop if more map registers
                     2031  6128 ;
```

```
                     2031  6129 ; Release the PCBB map registers
                     2031  6130 ;
      53   24 A5   D0  2031  6131          MOVL    UCB$L_CRB(R5),R3        ; Get CRB address
                     2035  6132          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                     2035  6133          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
         0162 C4     D0  2035  6134          MOVL    CDB_L_RINGMAP(R4),-     ; Setup map info in CRB
           34 A3         2039  6135                   CRB$L_INTD+VEC$W_MAPREG(R3) ;
                  0A 13  203B  6136          BEQL    50$                     ; Br if none
         0162 C4     D4  203D  6137          CLRL    CDB_L_RINGMAP(R4)       ; No more mapping info
                     2041  6138          RELMPR                          ; Release the map register
                     2047  6139 50$:      ;
                     2047  6140          ; Deallocate all receive buffers and complete all I/O request packets
                     2047  6141          ;
   56    00DC C4     9E  2047  6142          MOVAB   CDB_Q_QUEUES(R4),R6     ; Get address of first queue listhead
         57    06     3C  204C  6143          MOVZWL  #CDB_C_QUEUES,R7        ; Get number of queues
      53    00 B6     0F  204F  6144 60$:     REMQUE  @(R6),R3                ; Get next IRP/BUFFER
               79     1D  2053  6145          BVS     150$                    ; Br if none
      0A    0A A3     91  2055  6146          CMPB    IRP$B_TYPE(R3),S^#DYN$C_IRP ; Is this an IRP?
                  0A 13  2059  6147          BEQL    70$                     ; Br if yes
      1B    0A A3     91  205B  6148          CMPB    IRP$B_TYPE(R3),S^#DYN$C_CXB ; Is this a CXB?
                  09 13  205F  6149          BEQL    80$                     ; Br if yes
                     2061  6150          BUG_CHECK NOBUFPCKT,FATAL       ; Else, fatal error
                     2065  6151 ;
                     2065  6152 ; IRP
                     2065  6153 ;
            FB5A     30  2065  6154 70$:     BSBW    ABORT_PKT               ; Abort the IRP
              E5     11  2068  6155          BRB     60$                     ; Try for more
                     206A  6156 ;
                     206A  6157 ; CXB
                     206A  6158 ;
                     206A  6159 80$:     $DISPATCH       CXB$B_XQ_FUNC(R3),TYPE=B,-
                     206A  6160          <-      ;function       action
                     206A  6161
                     206A  6162                  <XQ_FC_V_XMIT   100$>,- ; XMIT request
                     206A  6163                  <XQ_FC_V_RECV   140$>,- ; RECV request
                     206A  6164                  <XQ_FC_V_INIT   100$>,- ; INIT request
                     206A  6165                  <XQ_FC_V_STOP   100$>,- ; STOP request
                     206A  6166                  <XQ_FC_V_CHMODE 100$>,- ; Change mode request
                     206A  6167          >
                     207D  6168
                     207D  6169          BUG_CHECK NOBUFPCKT,FATAL       ; Fatal error - not a valid IRP
                     2081  6170
                     2081  6171 ;
                     2081  6172 ; CXB - XMIT request
                     2081  6173 ;
                     2081  6174 100$:    ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
      16 24 A3     E8  2081  6175          BLBS    CXB$L_T_IRP(R3),120$    ; Br if not IRP address => FFI user
   53    24 A3     D0  2085  6176          MOVL    CXB$L_T_IRP(R3),R3      ; Else, get IRP address
50    0094 C3     D0  2089  6177          MOVL    IRP$L_XQ_SETUP(R3),R0   ; Get SETUP mode buffer
            06     13  208E  6178          BEQL    110$                    ; Br if not present
 00000000'GF       16  2090  6179          JSB     G^COM$DRVDEALMEM        ; Else, deallocate buffer
         FB29     30  2096  6180 110$:    BSBW    ABORT_PKT               ; Abort the IRP
           B4     11  2099  6181          BRB     60$                     ; Try for more
                     209B  6182
                     209B  6183 120$:    ;
                     209B  6184          ;  This is an FFI user, return CXB buffer
                     209B  6185          ;
```

```
                51    18 A3    3C   209B  6186           MOVZWL  CXB$W_BOFF(R3),R1        ; Get offset to start of data
                  51    53    C0   209F  6187           ADDL    R3,R1                   ; Compute start of Ethernet header
                51    0C A1    3C   20A2  6188           MOVZWL  XBUF_W_TYPE(R1),R1      ; Get protocol type
                   0C1D    30   20A6  6189           BSBW    MATCH_PROTYP            ; Find the protocol user
                   17 50    E9   20A9  6190           BLBC    R0,140$                 ; Br if no UCB, drop buffer
                      54    DD   20AC  6191           PUSHL   R4                      ; Save R4
            54    018D C5    D0   20AE  6192           MOVL    UCB$L_XQ_FFI(R5),R4     ; Get FFI block address
                      0B    13   20B3  6193           BEQL    130$                    ; Br if not there, drop buffer
                   50 2C    3C   20B5  6194           MOVZWL  #SS$_ABORT,R0           ; Set return status
                     14 B4    16   20B8  6195           JSB     @FFI$L_XMIT_DONE(R4)    ; Complete the XMIT
                      54 8ED0  20BB  6196           POPL    R4                      ; Restore R4
                      8F    11   20BE  6197           BRB     60$                     ; Try for more
                                 20C0  6198 130$:     ;
                                 20C0  6199           ;  No more FFI for CXB to complete
                                 20C0  6200           ;
                      54 8ED0  20C0  6201           POPL    R4                      ; Restore CDB address
                                 20C3  6202
                                 20C3  6203 ;
                                 20C3  6204 ; CXB - RECV request
                                 20C3  6205 ;
                                 20C3  6206
                   50    53    D0   20C3  6207 140$:     MOVL    R3,R0                   ; Copy CXB buffer address
             00000000'GF    16   20C6  6208           JSB     G^COM$DRVDEALMEM        ; Deallocate the buffer
                      81    11   20CC  6209 1060$:     BRB     60$                     ; Try for more
                                 20CE  6210           ;
                                 20CE  6211           ; Loop to next queue
                                 20CE  6212           ;
                   56    08    C0   20CE  6213 150$:     ADDL    #8,R6                   ; Skip to next queue listhead
                     F8 57    F5   20D1  6214           SOBGTR  R7,1060$                ; Loop if more queues
                                 20D4  6215           ;
                                 20D4  6216           ; Cleanup all I/O on all UNITS
                                 20D4  6217           ;
            55    0118 C4    D0   20D4  6218           MOVL    CDB_L_UCB0(R4),R5       ; Get UNIT #0 UCB address
                      0B    13   20D9  6219           BEQL    190$                    ; Br if none, yet
                55    30 A5    D0   20DB  6220 170$:     MOVL    UCB$L_LINK(R5),R5       ; Get next unit's address
                      05    13   20DF  6221           BEQL    190$                    ; Br if none
                         0026  30   20E1  6222           BSBW    SHUTDOWN                ; Shutdown the UNIT
                      F5    11   20E4  6223           BRB     170$                    ; Check if more UNIT's
                   00E8 8F    BA   20E6  6224 190$:     POPR    #^M<R3,R5,R6,R7>        ; Restore registers
                                 05   20EA  6225           RSB                             ; Return to caller
```

```
                     20EB  6227          .SBTTL SHUTDOWN - SHUT DOWN UNIT
                     20EB  6228          .SBTTL SHUTDOWN_PROTYP - SHUT DOWN PROTOCOL TYPE
                     20EB  6229 ;++
                     20EB  6230 ; SHUTDOWN - SHUT DOWN UNIT
                     20EB  6231 ; SHUTDOWN_PROTYP - SHUT DOWN PROTOCOL TYPE
                     20EB  6232 ;
                     20EB  6233 ; Functional description:
                     20EB  6234 ;
                     20EB  6235 ; This routine is used to shut down the XQ unit as a result of a
                     20EB  6236 ; SETMODE and SHUTDOWN. The action is to abort all I/O for the unit
                     20EB  6237 ; and then to clean up the unit data base.
                     20EB  6238 ;
                     20EB  6239 ; Inputs:
                     20EB  6240 ;
                     20EB  6241 ;       R3 = IRP address (SHUTDOWN_PROTYP entry only)
                     20EB  6242 ;       R4 = CDB address
                     20EB  6243 ;       R5 = UCB address
                     20EB  6244 ;
                     20EB  6245 ;       IPL = FIPL
                     20EB  6246 ;
                     20EB  6247 ; Outputs:
                     20EB  6248 ;
                     20EB  6249 ;       R3-R5 are preserved
                     20EB  6250 ;
                     20EB  6251 ;       R0-R2 are destroyed.
                     20EB  6252 ;--
                     20EB  6253 SHUTDOWN_PROTYP::                    ; Shut down protocol type
           04   E1    20EB  6254          BBC     #UCB$V_ONLINE,-     ; Br if not online
 19 64 A5              20ED  6255                  UCB$W_STS(R5),10$   ;
                     20F0  6256          ASSUME  UCB$V_XQ_INITED  EQ  0
 15 68 A5    E9      20F0  6257          BLBC    UCB$W_DEVSTS(R5),10$ ; Br if not inited
           03   E1    20F4  6258          BBC     #UCB$V_XQ_SHARE,-   ; Br if not a shared UCB
 11 68 A5              20F6  6259                  UCB$W_DEVSTS(R5),SHUTDOWN ; shutdown entire unit
                     20F9  6260 ;
                     20F9  6261 ; Try to find SHR data structure
                     20F9  6262 ;
          E60F   30  20F9  6263          BSBW    MATCH_SHR           ; Check PID and CHAN
            0B   12  20FC  6264          BNEQ    10$                 ; Br if NO MATCH, skip it
                     20FE  6265 ;
                     20FE  6266 ; Match found - clear inited bit and clean up all I/O on SHR data structure
                     20FE  6267 ;
            56   DD  20FE  6268          PUSHL   R6                  ; Save R6
       56   51   D0  2100  6269          MOVL    R1,R6               ; Copy SHR address
          0290   30  2103  6270          BSBW    CLEANUP_SHR         ; Cleanup the SHR data structure
            56 8ED0  2106  6271          POPL    R6                  ; Restore R6
                05  2109  6272 10$:     RSB                         ; Return to caller
                     210A  6273
                     210A  6274 SHUTDOWN::                           ; Shut down unit
           04   E1    210A  6275          BBC     #UCB$V_ONLINE,-
 04 64 A5              210C  6276                  UCB$W_STS(R5),5$    ; If BC not online
                     210F  6277
                     210F  6278          ASSUME  UCB$V_XQ_INITED  EQ  0
 01 68 A5    E8      210F  6279          BLBS    UCB$W_DEVSTS(R5),10$ ; Br if UCB is inited
                05  2113  6280 5$:      RSB                         ; It's not time to shut down, yet
                     2114  6281
                     2114  6282 ;
                     2114  6283 ; If a power failure occurred, and the protocol has both initialized the FFI
```

```
                        2114  6284 ; interface and supplied an asynchronous error routine, then call back
                        2114  6285 ; the protocol at this routine address with a status value indicating that
                        2114  6286 ; a power failure had taken place.
                        2114  6287 ;
                        2114  6288
            05   E1     2114  6289 10$:    BBC     #UCB$V_POWER,-          ; Skip notification if power failure
        19 64   A5      2116  6290                UCB$W_STS(R5),20$       ; did not occur
            54   DD     2119  6291         PUSHL   R4                      ; Save CDB address
    54   018D   C5   D0  211B  6292         MOVL    UCB$L_XQ_FFI(R5),R4     ; Retrieve FFI address
            0D   13     2120  6293         BEQL    15$                     ; Nothing to do if there isn't one
      52    1C   A4   D0  2122  6294         MOVL    FFI$L_ERROR(R4),R2      ; Retrieve asynch error routine address
            07   13     2126  6295         BEQL    15$                     ; Nothing to do if there isn't one
    50  0364   8F   3C  2128  6296         MOVZWL  #SS$_POWERFAIL,R0       ; Indicate that a powerfailure occurred
            62   16     212D  6297         JSB     (R2)                    ; Call back the asynch error routine
          54 8ED0        212F  6298 15$:    POPL    R4                      ; Restore CDB address
                        2132  6299
                        2132  6300 ;
                        2132  6301 ; Start a 3-second timer to restart any UNIT needing automatic restart.
                        2132  6302 ; This restart timer only runs if the device was halted due to a fatal error.
                        2132  6303 ;
                        2132  6304 ; Note, that the UCB multicast address list is purged, which will nullify
                        2132  6305 ; any restart operation that may be performed by the QNA driver itself (only
                        2132  6306 ; if the user has specified any multicast addresses).
                        2132  6307 ;
                        2132  6308
        00C8   8F   BB  2132  6309 20$:    PUSHR   #^M<R3,R6,R7>           ; Save registers
            0F   E1     2136  6310         BBC     #UCB$V_XQ_RESTART,-     ; Br if this UNIT does not need
        43 68   A5      2138  6311                UCB$W_DEVSTS(R5),22$    ;  automatic restart
                        213B  6312         ASSUME  IRP$C_LENGTH GE TQE$C_LENGTH
      51    F4   8F   9A  213B  6313         MOVZBL  #IRP$C_LENGTH+TQE$C_LENGTH,R1 ; Get size of IRP/TQE
  00000000'GF      16  213F  6314         JSB     G^EXE$ALONONPAGED       ; Try to allocate a IRP/TQE
            36   50   E9  2145  6315         BLBC    R0,22$                  ; Br if failure - too bad
                        2148  6316         ASSUME  IRP$Q_STATION GT TQE$C_LENGTH
          4000   8F   A8  2148  6317         BISW    #UCB$M_XQ_INTERLOCK,-   ; Interlock the RESTART bit
            68   A5      214C  6318                UCB$W_DEVSTS(R5)        ;
                        214E  6319
                        214E  6320         ASSUME  TQE$B_TYPE EQ TQE$W_SIZE+2
                        214E  6321         ASSUME  TQE$B_RQTYPE EQ TQE$B_TYPE+1
      000F00F4   8F   D0  214E  6322         MOVL    #<<DYN$C_TQE@16>!<IRP$C_LENGTH+TQE$C_LENGTH>>,- ; Set STRUCTURE
            08   A2      2154  6323                TQE$W_SIZE(R2)          ;  TYPE and SIZE
      00C8   C2   55   D0  2156  6324         MOVL    R5,TQE$C_LENGTH+IRP$L_RBOFF(R2) ; Save UCB address in IRP
                        215B  6325         PUSHQ   R4                      ; Save R4, R5
          55   52   D0  215E  6326         MOVL    R2,R5                   ; Copy TQE address
00000000 01C9C380   8F   7D  2161  6327         MOVQ    #RESTART_DELTA,-        ; Set the delta time
            20   A5      216B  6328                TQE$Q_DELTA(R5)         ;
        53  FCE9   CF   9E  216D  6329         MOVAB   W^RESTART_ROUT,R3       ; Get address of RESTART routine
            01   90     2172  6330         MOVB    #TQE$C_SSSNGL,-         ; Set the request type
            2C   A5      2174  6331                TQE$L_RQPID(R5)         ;
          F1B3   30     2176  6332         BSBW    FORK_TIMER              ; FORK to startup the timer
                        2179  6333         POPQ    R4                      ; Restore R4, R5
            05   11     217C  6334         BRB     23$                     ; Contine
                        217E  6335
                        217E  6336 22$:    CLRBIT  #UCB$V_XQ_RESTART,-     ; Restart is not possible!
                        217E  6337                UCB$W_DEVSTS(R5)
  00D2   C5   2328   8F   B0  2183  6338 23$:    MOVW    #INIT_C_QUOTA,UCB$W_XQ_HBQ(R5) ; Reset hardware buffer quota
          64   A5   23   AA  218A  6339         BICW    #UCB$M_INT!UCB$M_POWER!-
                        218E  6340                UCB$M_TIM,UCB$W_STS(R5) ; Reset device status
```

```
                68 A5    11    AA  218E  6341          BICW    #UCB$M_XQ_INITED!-       ; No longer inited
                                   2192  6342                   UCB$M_XQ_RUN,UCB$W_DEVSTS(R5) ;  ..or  running
                                   2192  6343          CLRBIT  #XM$V_STS_ACTIVE,UCB$L_DEVDEPEND(R5) ; Clear active bit
          05 0114 C4     10    E1  2197  6344          BBC     #XM$V_ERR_FATAL,CDB_L_DEVDEPEND(R4),25$ ; Br if not FATAL
                                   219D  6345          SETBIT  #XM$V_ERR_FATAL,UCB$L_DEVDEPEND(R5) ; Else, indicate FATAL
                                   21A2  6346 ;
                                   21A2  6347 ; Reset UCB multicast address list
                                   21A2  6348 ;
             0214 C4     55    D1  21A2  6349 25$:     CMPL    R5,CDB_L_PRMUSER(R4)     ; Is this unit the PROMISCUOUS user?
                         0C    12  21A7  6350          BNEQ    27$                      ; Br if not
                   0214 C4     D4  21A9  6351          CLRL    CDB_L_PRMUSER(R4)        ; Else, clear the PROMISCUOUS user addr
                         01    90  21AD  6352          MOVB    #NMA$C_STATE_OFF,-       ; Don't forget about the CDB
                   024B C4         21AF  6353                  CDB_B_PRM(R4)            ;  parameter
                      0154     30  21B2  6354          BSBW    BLD_STOP_IRP             ; Build an IRP to RESET hardware mode
                         0F    E0  21B5  6355 27$:     BBS     #UCB$V_XQ_RESTART,-       ; Br if this UNIT is restarting
                16 68 A5         21B7  6356                  UCB$W_DEVSTS(R5),28$     ;  don't clear multicast list
                                   21BA  6357          PUSHQ   R4                       ; Save R4, R5
                   00E5 C5   94  21BD  6358          CLRB    UCB$B_XQ_MULTI(R5)       ; No more multicast addresses
          00   00E7 C5   00    2C  21C1  6359          MOVC5   #0,UCB$G_XQ_MULTI(R5),#0,- ; Zero the structure
          00E7 C5   0048 8F         21C7  6360                  #6*MAX_C_MLT,UCB$G_XQ_MULTI(R5) ;
                                   21CD  6361          POPQ    R4                       ; Restore R4, R5
                                   21D0  6362 ;
                                   21D0  6363 ; Reset CDB multicast address list, Flush all attention ASTs.
                                   21D0  6364 ;
                      04DE     30  21D0  6365 28$:     BSBW    ADD_MULTI                ; Re-calculate multicast address list
                         54    DD  21D3  6366          PUSHL   R4                       ; Save CDB address
             57   00C0 C5   9E    21D5  6367 30$:     MOVAB   UCB$L_XQ_AST(R5),R7      ; Get address of AST listhead
                  50   67   D0    21DA  6368          MOVL    (R7),R0                  ; Anything in list?
                         1B    13  21DD  6369          BEQL    40$                      ; Br if not
                    56   22 A0   3C  21DF  6370          MOVZWL  ACB$L_KAST+10(R0),R6     ; Force channel match
                    52   24 A0   3C  21E3  6371          MOVZWL  ACB$L_KAST+12(R0),R2     ; Get process index
          54   00000000'GF   D0    21E7  6372          MOVL    G^SCH$GL_PCBVEC,R4       ; Get PCB address vector address
                    54   6442   D0  21EE  6373          MOVL    (R4)[R2],R4              ; Get PCB address
                00000000'GF   16    21F2  6374          JSB     G^COM$FLUSHATTNS         ; Flush AST
                         DB    11  21F8  6375          BRB     30$
                   54 8ED0     21FA  6376 40$:     POPL    R4                       ; Restore CDB address
                               21FD  6377 ;
                               21FD  6378 ; Complete all RCV IRPs for this unit
                               21FD  6379 ;
                               21FD  6380          ASSUME  UCB$C_XQ_QUEUES-1 EQ 3   ; One queue for shared users
             53   00A8 D5     0F    21FD  6381 45$:     REMQUE  @UCB$Q_XQ_RCVREQ(R5),R3 ; Get IRP
                         05    1D  2202  6382          BVS     50$                      ; Br if none
                      F9BB     30  2204  6383          BSBW    ABORT_PKT                ; Abort the I/O request
                         F4    11  2207  6384          BRB     45$                      ; Get next IRP
                               2209  6385 ;
                               2209  6386 ; Complete all XMIT CXBs for this unit
                               2209  6387 ;
             53   00B0 D5     0F    2209  6388 50$:     REMQUE  @UCB$Q_XQ_XMTREQ(R5),R3 ; Get CXB
                         2C    1D  220E  6389          BVS     55$                      ; Br if none
                               2210  6390          ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
                12 24 A3     E9    2210  6391          BLBC    CXB$L_T_IRP(R3),53$      ; Br if IRP address
                         54    DD  2214  6392          PUSHL   R4                       ; Save CDB address
                54   018D C5   D0    2216  6393          MOVL    UCB$L_XQ_FFI(R5),R4      ; Get FFI block address
                   50   2C     3C    221B  6394          MOVZWL  #SS$_ABORT,R0            ; Set status return
                      14 B4     16    221E  6395          JSB     @FFI$L_XMIT_DONE(R4)     ; Complete CXB
                   54 8ED0     2221  6396          POPL    R4                       ; Restore CDB address
                         E3    11  2224  6397          BRB     50$                      ; Get next CXB
```

```
                                2226  6398
          53    24 A3   D0      2226  6399 53$:    MOVL    CXB$L_T_IRP(R3),R3      ; Get IRP address
        50    0094 C3   D0      222A  6400          MOVL    IRP$L_XQ_SETUP(R3),R0   ; Get SETUP mode buffer
                06      13      222F  6401          BEQL    54$                     ; Br if none
         00000000'GF    16      2231  6402          JSB     G^COM$DRVDEALMEM        ; Else, deallocate the buffer
                F988    30      2237  6403 54$:    BSBW    ABORT_PKT               ; Abort the I/O request
                  CD    11      223A  6404          BRB     50$                     ; Get next CXB
                        223C  6405 ;
                        223C  6406 ; Deallocate all receive CXBs
                        223C  6407 ;
        52    00A0 D5   0F      223C  6408 55$:    REMQUE  @UCB$Q_XQ_RCVMSG(R5),R2 ; Get message buffer
                  24    1D      2241  6409          BVS     70$                     ; Br if none
                42 A5   A0      2243  6410          ADDW    UCB$W_DEVBUFSIZ(R5),-   ; Restore quota
                00C8 C5         2246  6411                  UCB$W_XQ_QUOTA(R5)      ;
                        2249  6412 ;
                        2249  6413 ;       The buffer may be smaller than the normal message size, if this
                        2249  6414 ;       is a cloned buffer for the promiscuous user. Therefore, we must
                        2249  6415 ;       check to make sure the buffer is large enough to be returned
                        2249  6416 ;       the the device's receive buffer pool.
                        2249  6417 ;
                    A1  2249  6418          ADDW3   #CXB$C_HEADER+-         ; Calculate size of "normal"
                        224A  6419                  CXB$C_TRAILER,-         ;  receive buffer
  50    0110 C4   004C 8F       224A  6420                  CDB_W_BSZ(R4),R0        ;
              08 A2   50    B1  2251  6421          CMPW    R0,CXB$W_SIZE(R2)       ; Can buffer be returned?
                  05    12      2255  6422          BNEQ    60$                     ; Br if not, delete buffer instead
                F1E6    30      2257  6423          BSBW    ADDRCVLIST              ; Try to add to receiver list
                  AD    11      225A  6424          BRB     50$                     ; Loop for more
                        225C  6425
          50    52      D0      225C  6426 60$:    MOVL    R2,R0                   ; Copy buffer address for deallocation
         00000000'GF    16      225F  6427          JSB     G^COM$DRVDEALMEM        ; Deallocate the buffer
                  A2    11      2265  6428          BRB     50$                     ; Loop for more
                        2267  6429 ;
                        2267  6430 ; Cleanup all SHR structures if fatal error
                        2267  6431 ;
                03      E1      2267  6432 70$:    BBC     #UCB$V_XQ_SHARE,-       ; Br if not a SHARED UCB
             30 68 A5           2269  6433                  UCB$W_DEVSTS(R5),100$   ;
        56    00C4 C5   D0      226C  6434          MOVL    UCB$L_XQ_DEFUSR(R5),R6  ; Get default SHR structure address
                  0B    13      2271  6435          BEQL    90$                     ; Br if none
                0120    30      2273  6436          BSBW    CLEANUP_SHR             ; Else, cleanup the structure
                  10    E1      2276  6437          BBC     #XM$V_ERR_FATAL,-       ; Br if not a fatal error
             03 44 A5           2278  6438                  UCB$L_DEVDEPEND(R5),90$
                018C    30      227B  6439          BSBW    DELETE_SHR              ; And delete the structure
        56    0098 C5   D0      227E  6440 90$:    MOVL    UCB$Q_XQ_SHARE(R5),R6   ; Get address of next LIMITED user
  0098 C5       56    D1      2283  6441          CMPL    R6,UCB$Q_XQ_SHARE(R5)   ; End of list?
                  45    13      2288  6442          BEQL    120$                    ; Br if yes, don't restore quota (yet)
                0109    30      228A  6443          BSBW    CLEANUP_SHR             ; Cleanup the I/O
                  10    E1      228D  6444          BBC     #XM$V_ERR_FATAL,-       ; Br if not a fatal error
             EC 44 A5           228F  6445                  UCB$L_DEVDEPEND(R5),90$
                  0F    E0      2292  6446          BBS     #UCB$V_XQ_RESTART,-     ; Br if this UNIT is re-starting
             E7 68 A5           2294  6447                  UCB$W_DEVSTS(R5),90$
                0170    30      2297  6448          BSBW    DELETE_SHR              ; Else, delete the structure
                  E2    11      229A  6449          BRB     90$                     ; Look for more
                        229C  6450 ;
                        229C  6451 ; Restore quota
                        229C  6452 ;
                  0F    E0      229C  6453 100$:   BBS     #UCB$V_XQ_RESTART,-     ; Br if this UNIT is re-starting
             5A 68 A5           229E  6454                  UCB$W_DEVSTS(R5),140$
```

```
                50   00B8 C5    3C   22A1  6455          MOVZWL  UCB$L_XQ_PID(R5),R0      ; Get PID of last starter
            51   00000000'GF    D0   22A6  6456          MOVL    G^SCH$GL_PCBVEC,R1      ; Address PCB vector
                    50   6140   D0   22AD  6457          MOVL    (R1)[R0],R0             ; Get PCB of owner
                         60 A0  D1   22B1  6458          CMPL    PCB$L_PID(R0),-
                      00B8 C5               22B4  6459                  UCB$L_XQ_PID(R5)        ; Still there?
                           16  12   22B7  6460          BNEQ    120$                    ; If NEQ no
                50   0080 C0    D0   22B9  6461          MOVL    PCB$L_JIB(R0),R0        ; Get JIB address
                51   00C8 C5    3C   22BE  6462          MOVZWL  UCB$W_XQ_QUOTA(R5),R1   ; Convert to longword
                  20 A0   51    C0   22C3  6463          ADDL    R1,JIB$L_BYTCNT(R0)     ; Return byte count quota
                  24 A0   51    C0   22C7  6464          ADDL    R1,JIB$L_BYTLM(R0)      ; ..and byte limit quota
                      00C8 C5   B4   22CB  6465          CLRW    UCB$W_XQ_QUOTA(R5)      ; Prevent this from being
                                     22CF  6466                                          ; returned again
                                     22CF  6467 ;
                                     22CF  6468 ; Delete the STARTUP IRP for point-to-point mode
                                     22CF  6469 ;
                50   0191 C5    D0   22CF  6470 120$:   MOVL    UCB$L_XQ_STIRP(R5),R0   ;% Get the startup IRP address
                           0A   13   22D4  6471          BEQL    130$                    ;% Br if none
                      0191 C5   D4   22D6  6472          CLRL    UCB$L_XQ_STIRP(R5)      ;% All done
                00000000'GF     16   22DA  6473          JSB     G^COM$DRVDEALMEM        ;% Deallocate the IRP
                                     22E0  6474 ;
                                     22E0  6475 ; If there is an FFI block and the SHUT_DONE routine is set, then
                                     22E0  6476 ; notify the FFI user that shutdown is now complete.
                                     22E0  6477 ;
                50   018D C5    D0   22E0  6478 130$:   MOVL    UCB$L_XQ_FFI(R5),R0     ; Get FFI block address
                           14   13   22E5  6479          BEQL    140$                    ; Br if none
                      018D C5   D4   22E7  6480          CLRL    UCB$L_XQ_FFI(R5)        ; Cleanup FFI interface
                  51   20 A0   D0   22EB  6481          MOVL    FFI$L_SHUT_DONE(R0),R1  ; Get address of routine
                           0A   13   22EF  6482          BEQL    140$                    ; Br if none
                           54   DD   22F1  6483          PUSHL   R4                      ; Save CDB address
                      54   50   D0   22F3  6484          MOVL    R0,R4                   ; Copy FFI block address
                           61   16   22F6  6485          JSB     (R1)                    ; Call back FFI user
                           54 8ED0   22F8  6486          POPL    R4                      ; Restore CDB address
                                     22FB  6487 ;
                                     22FB  6488 ; Decrement UNIT count on CDB and cleanup CDB if last unit
                                     22FB  6489 ;
020F C4    97   22FB  6490 140$:   DECB    CDB_B_UNTCNT(R4)        ; One less unit on CDB
      03   12   22FF  6491          BNEQ    150$                    ; Br if more
  FCD2     30   2301  6492          BSBW    SHUTDOWN_QNA            ; Else, shutdown entire QNA
00C8 8F    BA   2304  6493 150$:   POPR    #^M<R3,R6,R7>           ; Restore registers
           05   2308  6494          RSB                             ; Return to caller
                2309  6495
```

```
                     2309  6497          .SBTTL  BLD_STOP_IRP - Build an IRP to reset promiscuous mode
                     2309  6498 ;++
                     2309  6499 ; BLD_STOP_IRP - Build an IRP to reset the promiscuous mode
                     2309  6500 ;
                     2309  6501 ; Functional description:
                     2309  6502 ;
                     2309  6503 ; This routine will allocate and build an IRP to reset the hardware mode
                     2309  6504 ; from promiscous.
                     2309  6505 ;
                     2309  6506 ; Inputs:
                     2309  6507 ;       R4 = CDB address
                     2309  6508 ;       R5 = UCB address
                     2309  6509 ;
                     2309  6510 ; Outputs:
                     2309  6511 ;       R0,R1,R2,R3 are destroyed.
                     2309  6512 ;--
                     2309  6513          .ENABL  LSB
                     2309  6514 BLD_STOP_IRP:                                ; Build an IRP to reset hardware mode
                     2309  6515 ;
                     2309  6516 ; NOTE - we must use EXE$ALONONPAGED to allocate the IRP because the other
                     2309  6517 ;       routines reset the IPL to ASTDEL.
                     2309  6518 ;
          10   E0    2309  6519          BBS     #XM$V_ERR_FATAL,-       ; Br if fatal error,
 0E 0114 C4          230B  6520                  CDB_L_DEVDEPEND(R4),10$ ;  ignore reset of mode
51   00C4 8F   3C    230F  6521          MOVZWL  #IRP$C_LENGTH,R1        ; Set length of IRP
 00000000'GF   16    2314  6522          JSB     G^EXE$ALONONPAGED       ; Try to allocate an IRP
       01 50   E8    231A  6523          BLBS    R0,20$                  ; Okay if buffer allocated
               05    231D  6524 10$:     RSB                             ; Else, too bad if we can't do it
                     231E  6525
 08 A2    51   B0    231E  6526 20$:     MOVW    R1,IRP$W_SIZE(R2)       ; Fill in the size field
             5C   10  2322  6527          BSBB    BLD_IRP                 ; Build a template IRP
0C A3   39'AF   9E    2324  6528          MOVAB   B^DELETE_BLOCK,IRP$L_PID(R3) ; Store return address from IOPOST
          F043  30    2329  6529          BSBW    SETUP_MODE              ; Allocate setup mode buffer
       05 50   E9    232C  6530          BLBC    R0,70$                  ; Leave on error
          03   90    232F  6531          MOVB    #XQ_FC_V_STOP,-         ; Set function request,
       20 A2         2331  6532                  CXB$B_XQ_FUNC(R2)       ;  looks like a STOP
               05    2333  6533          RSB                             ; Return to queue request to DEQNA
                     2334  6534
       50   53   D0  2334  6535 70$:     MOVL    R3,R0                   ; Copy IRP address
             03   11  2337  6536          BRB     90$                     ; Deallocate IRP
                     2339  6537
                     2339  6538 DELETE_BLOCK:                            ; Deallocate a data structure
       50   55   D0  2339  6539          MOVL    R5,R0                   ; Get address of structure
 00000000'GF   17    233C  6540 90$:     JMP     G^COM$DRVDEALMEM        ; Deallocate the structure
                     2342  6541
                     2342  6542          .DSABL  LSB
```

```
                                2342  6544          .SBTTL  BLD_STRT_IRP - Build a point-to-point startup IRP
                                2342  6545 ;++
                                2342  6546 ; BLD_STRT_IRP - Build a point-to-point startup IRP
                                2342  6547 ;
                                2342  6548 ; Functional description:
                                2342  6549 ;
                                2342  6550 ; This routine will build an IRP to perform a datalink startup with a
                                2342  6551 ; remote system.
                                2342  6552 ;
                                2342  6553 ; Inputs:
                                2342  6554 ;       R2 = IRP address
                                2342  6555 ;       R4 = CDB address
                                2342  6556 ;       R5 = UCB address
                                2342  6557 ;
                                2342  6558 ; Outputs:
                                2342  6559 ;       R0,R1,R2,R3 are destroyed.
                                2342  6560 ;--
                                2342  6561 BLD_STRT_IRP:                           ;% Build a startup IRP
                                2342  6562 ;
                                2342  6563 ; We will use the back part of the IRP to build the data message. The
                                2342  6564 ; data message only contains the standard header plus one byte of ^XAA.
                                2342  6565 ;
                                2342  6566          ASSUME  IRP$C_XQ_STD+17 LE IRP$C_LENGTH ;% 14 bytes of header +
                                2342  6567                                                  ;%    2 bytes of count + msg type.
                          3C    10  2342  6568          BSBB    BLD_IRP                 ;% Build the IRP
                    52    60 A3   9E  2344  6569          MOVAB   IRP$C_XQ_STD(R3),R2     ;% Point to data portion of IRP
                       00CC C5    7D  2348  6570          MOVQ    UCB$G_XQ_DES(R5),-      ;% Store destination address
                            62        234C  6571                  XBUF_G_DEST(R2)         ;%
                0C  A2    0660 8F    B0  234D  6572          MOVW    #XQ_C_STPRO,XBUF_W_TYPE(R2) ;% Store protocol type
                   0E  A2    01    B0  2353  6573          MOVW    #1,XBUF_W_SIZE(R2)      ;% Store message size
                 10  A2     AA 8F    90  2357  6574          MOVB    #^XAA,XBUF_W_SIZE+2(R2) ;% Transmit one start byte of data
                 05  68 A5    05    E0  235C  6575          BBS     #UCB$V_XQ_START,UCB$W_DEVSTS(R5),50$ ;% Br if start
                 10  A2     AB 8F    90  2361  6576          MOVB    #^XAB,XBUF_W_SIZE+2(R2) ;% Transmit one stack byte of data
                 0C  A3      7F'AF   9E  2366  6577 50$:     MOVAB   B^90$,IRP$L_PID(R3)     ;% Store return address
                   21  A3    01    90  236B  6578          MOVB    #XQ_FC_V_XMIT,IRP$B_XQ_FUNC(R3) ;% Set function request
                                     236F  6579          ASSUME  XBUF_C_HEADER EQ XBUF_W_SIZE
                             11    B0  236F  6580          MOVW    #XBUF_C_HEADER+3,-      ;% Set data size
                          32  A3         2371  6581                  IRP$W_BCNT(R3)          ;%
                    3C  A3    52    D0  2373  6582          MOVL    R2,IRP$L_XQ_SYSBUF(R3)  ;% Set buffer address
                 00E0  D4    63    0E  2377  6583          INSQUE  (R3),@CDB_Q_XMTREQ+4(R4);% Insert transmit request
                        E114    30    237C  6584          BSBW    XMT_ALT_START           ;% Startup transmit process
                                05    237F  6585 90$:     RSB                             ;% Return to caller
                                      2380  6586
```

```
                                                I  1
                2380  6588          .SBTTL  BLD_IRP - Build an IRP routine
                2380  6589 ;++
                2380  6590 ; BLD_IRP - Build an IRP routine
                2380  6591 ;
                2380  6592 ; Functional description:
                2380  6593 ;
                2380  6594 ; This routine will build a simple IRP and allow the caller to fill in the
                2380  6595 ; function requested and then queue it to the DEQNA.
                2380  6596 ;
                2380  6597 ; Inputs:
                2380  6598 ;       R2 = IRP address
                2380  6599 ;       R5 = UCB address
                2380  6600 ;
                2380  6601 ; Outputs:
                2380  6602 ;       R3 = IRP address
                2380  6603 ;       R0-R2 are destroyed.
                2380  6604 ;       R4,R5 are preserved.
                2380  6605 ;--
                2380  6606 BLD_IRP:                                       ; Build an IRP
53   82   7E    2380  6607          MOVAQ   (R2)+,R3                     ; Save IRP address, skip to size field
                2383  6608          ASSUME  IRP$W_SIZE EQ 8
                2383  6609          ASSUME  IRP$B_TYPE EQ IRP$W_SIZE+2
                2383  6610          ASSUME  IRP$B_RMOD EQ IRP$B_TYPE+1
     82   B5    2383  6611          TSTW    (R2)+                        ; Skip SIZE
82   0A   B0    2385  6612          MOVW    #DYN$C_IRP,(R2)+             ; Make it look like an IRP
                2388  6613          ASSUME  IRP$L_PID EQ IRP$B_RMOD+1
                2388  6614          ASSUME  IRP$L_AST EQ IRP$L_PID+4
     82   7C    2388  6615          CLRQ    (R2)+                        ; Clear PID, AST
                238A  6616          ASSUME  IRP$L_ASTPRM EQ IRP$L_AST+4
                238A  6617          ASSUME  IRP$L_WIND EQ IRP$L_ASTPRM+4
     82   7C    238A  6618          CLRQ    (R2)+                        ; Clear ASTPRM, WIND
                238C  6619          ASSUME  IRP$L_UCB EQ IRP$L_WIND+4
82   55   D0    238C  6620          MOVL    R5,(R2)+                     ; Store UCB address
                238F  6621          ASSUME  IRP$W_FUNC EQ IRP$L_UCB+4
                238F  6622          ASSUME  IRP$B_EFN EQ IRP$W_FUNC+2
                238F  6623          ASSUME  IRP$B_PRI EQ IRP$B_EFN+1
                238F  6624          ASSUME  IRP$L_IOSB EQ IRP$B_PRI+1
     82   7C    238F  6625          CLRQ    (R2)+                        ; Clear FUNC, EFN, PRI, IOSB
                2391  6626          ASSUME  IRP$W_CHAN EQ IRP$L_IOSB+4
                2391  6627          ASSUME  IRP$W_STS EQ IRP$W_CHAN+2
                2391  6628          ASSUME  IRP$L_SVAPTE EQ IRP$W_STS+2
     82   7C    2391  6629          CLRQ    (R2)+                        ; Clear CHAN, STS, SVAPTE
                2393  6630          ASSUME  IRP$W_BOFF EQ IRP$L_SVAPTE+4
                2393  6631          ASSUME  IRP$W_BCNT EQ IRP$W_BOFF+2
                2393  6632          ASSUME  IRP$L_BCNT EQ IRP$W_BCNT
     82   7C    2393  6633          CLRQ    (R2)+                        ; Clear BOFF, BCNT
          05    2395  6634          RSB                                  ; Return to caller
                2396  6635
```

```
                        2396  6637          .SBTTL  CLEANUP_SHR - CLEANUP ALL I/O ON SHARE DATA STRUCTURE
                        2396  6638          .SBTTL  DELETE_SHR - DELETE SHR DATA STRUCTURE
                        2396  6639 ;++
                        2396  6640 ; CLEANUP_SHR - CLEANUP ALL I/O ON SHARE DATA STRUCTURE
                        2396  6641 ;
                        2396  6642 ; This routine aborts all read request in progress and return all message
                        2396  6643 ; buffers back to the CDB structure for re-use.
                        2396  6644 ;
                        2396  6645 ; Inputs:
                        2396  6646 ;
                        2396  6647 ;       R4 = CDB address
                        2396  6648 ;       R5 = UCB address
                        2396  6649 ;       R6 = SHR address
                        2396  6650 ;
                        2396  6651 ;       IPL = FIPL
                        2396  6652 ;
                        2396  6653 ; Outputs:
                        2396  6654 ;
                        2396  6655 ;       R0-R2 are destroyed.
                        2396  6656 ;       All other registers are preserved.
                        2396  6657 ;--
                        2396  6658 CLEANUP_SHR::                                ; Cleanup all I/O on SHR structure
    53   DD             2396  6659          PUSHL   R3                          ; Save R3
                        2398  6660
                        2398  6661          .IF DF  POINT
    57   DD             2398  6662          PUSHL   R7                          ;% Save R7
                        239A  6663 ;
                        239A  6664 ; Complete all waiting transmit IRPs
                        239A  6665 ;
    57   00B0 C5   9E   239A  6666 1$:      MOVAB   UCB$Q_XQ_XMTREQ(R5),R7      ;% Get address of XMIT wait queue
         53   67   D0   239F  6667          MOVL    (R7),R3                     ;% Travel queue
         57   53   D1   23A2  6668 2$:      CMPL    R3,R7                       ;% At end of queue?
              40   13   23A5  6669          BEQL    9$                          ;% Br if yes
      50   24   A3   D0   23A7  6670          MOVL    CXB$L_T_IRP(R3),R0          ;% Get (presumed) IRP address
                        23AB  6671          ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
              07   50   E8   23AB  6672          BLBS    R0,4$                       ;% Br if not IRP address, only one chan
    10 A6     28   A0   B1   23AE  6673          CMPW    IRP$W_CHAN(R0),SHR_W_CHAN(R6) ;% Same as SHR chan?
              2D   12   23B3  6674          BNEQ    8$                          ;% Br if not, else
         53   63   0F   23B5  6675 4$:      REMQUE  (R3),R3                     ;% Remove CXB from list
           15   50   E8   23B8  6676          BLBS    R0,6$                       ;% Br if FAST interface
         53   50   D0   23BB  6677          MOVL    R0,R3                       ;% Else, copy IRP address
    50   0094 C3   D0   23BE  6678          MOVL    IRP$L_XQ_SETUP(R3),R0       ;% Get SETUP mode buffer
              06   13   23C3  6679          BEQL    5$                          ;% Br if none
     00000000'GF   16   23C5  6680          JSB     G^COM$DRVDEALMEM            ;% Else, deallocate the buffer
           F7F4   30   23CB  6681 5$:      BSBW    ABORT_PKT                   ;% Abort the I/O request
              CA   11   23CE  6682          BRB     1$                          ;% Look for more
              54   DD   23D0  6683 6$:      PUSHL   R4                          ;% Save R4
         50   2C   3C   23D2  6684          MOVZWL  #SS$_ABORT,R0               ;% Setup error return
    54   018D C5   D0   23D5  6685          MOVL    UCB$L_XQ_FFI(R5),R4         ;% Get FFI block address
           14   B4   16   23DA  6686          JSB     @FFI$L_XMIT_DONE(R4)        ;% Complete the XMIT CXB
              54 8ED0   23DD  6687          POPL    R4                          ;% Restore R4
              B8   11   23E0  6688          BRB     1$                          ;% Look for more
         53   63   D0   23E2  6689 8$:      MOVL    (R3),R3                     ;% Travel link
              E4   11   23E5  6690          BRB     5$                          ;% Check for end of queue
              57 8ED0   23E7  6691 9$:      POPL    R7                          ;% Restore R7
                        23EA  6692          .ENDC
                        23EA  6693
```

```
                        23EA  6694          ASSUME  SHR_C_QUEUES EQ 2
                        23EA  6695 ;
                        23EA  6696 ; Complete all IRPs for this structure
                        23EA  6697 ;
53   20 B6    0F        23EA  6698 10$:     REMQUE  @SHR_Q_RCVREQ(R6),R3    ; Get IRP
       05     1D        23EE  6699          BVS     20$                     ; Br if none
     F7CF     30        23F0  6700          BSBW    ABORT_PKT               ; Abort the I/O request
       F5     11        23F3  6701          BRB     10$                     ; Get next IRP
                        23F5  6702 ;
                        23F5  6703 ; Deallocate all message blocks
                        23F5  6704 ;
52   18 B6    0F        23F5  6705 20$:     REMQUE  @SHR_Q_RCVMSG(R6),R2    ; Get message buffer
       0B     1D        23F9  6706          BVS     30$                     ; Br if none
     42 A5    A0        23FB  6707          ADDW    UCB$W_DEVBUFSIZ(R5),-   ; Restore quota
   00C8 C5              23FE  6708           UCB$W_XQ_QUOTA(R5)             ;
     F03C     30        2401  6709          BSBW    ADDRCVLIST              ; Try to add to receiver list
       EF     11        2404  6710          BRB     20$                     ; Loop
       53  8ED0        2406  6711 30$:     POPL    R3                      ; Restore R3
       05     2409  6712          RSB                             ; Return to caller
                        240A  6713
                        240A  6714 ;++
                        240A  6715 ; DELETE_SHR - DELETE SHARE DATA STRUCTURE
                        240A  6716 ;
                        240A  6717 ; This routine deallocates the SHR data structure to system pool.
                        240A  6718 ;
                        240A  6719 ; Inputs:
                        240A  6720 ;
                        240A  6721 ;       R5 = UCB address
                        240A  6722 ;       R6 = SHR address
                        240A  6723 ;
                        240A  6724 ;       IPL = FIPL
                        240A  6725 ;
                        240A  6726 ; Outputs:
                        240A  6727 ;
                        240A  6728 ;       R0-R1 are destroyed.
                        240A  6729 ;       All other registers are preserved.
                        240A  6730 ;--
                        240A  6731 DELETE_SHR::                         ; Delete SHR data structure
     5C A5    B7        240A  6732          DECW    UCB$W_REFC(R5)          ; One less user of the unit
 00C4 C5      56  D1    240D  6733          CMPL    R6,UCB$L_XQ_DEFUSR(R5)  ; Is this the default user?
       1C     13        2412  6734          BEQL    30$                     ; Br if yes
 51    0098 C5  9E      2414  6735          MOVAB   UCB$Q_XQ_SHARE(R5),R1   ; Get address of SHARE queue
       50     61  D0    2419  6736          MOVL    (R1),R0                 ; Get address of next in queue
       51     50  D1    241C  6737 10$:     CMPL    R0,R1                   ; Back to front of list?
              4D  13    241F  6738          BEQL    90$                     ; Br if none found
       50     56  D1    2421  6739          CMPL    R6,R0                   ; Is this the one?
              05  13    2424  6740          BEQL    20$                     ; Br if yes
       50     60  D0    2426  6741          MOVL    (R0),R0                 ; Else, get next in queue
              F1  11    2429  6742          BRB     10$                     ; And try for match
       50     60  0F    242B  6743 20$:     REMQUE  (R0),R0                 ; Remove structure from list
              04  11    242E  6744          BRB     40$                     ; And delete the structure
     00C4 C5      D4    2430  6745 30$:     CLRL    UCB$L_XQ_DEFUSR(R5)     ; No more default user
   50    0C A6    3C    2434  6746 40$:     MOVZWL  SHR_L_PID(R6),R0        ; Get PID SHR structure
51   00000000'GF  D0    2438  6747          MOVL    G^SCH$GL_PCBVEC,R1      ; Address PCB vector
     50    6140  D0    243F  6748          MOVL    (R1)[R0],R0             ; Get PCB of owner
           60 A0  D1    2443  6749          CMPL    PCB$L_PID(R0),-
           0C A6        2446  6750                  SHR_L_PID(R6)           ; Still there?
```

```
                 1B  12  2448  6751            BNEQ    60$                             ; If NEQ no
50    0080 C0    D0  244A  6752                MOVL    PCB$L_JIB(R0),R0                ; Get JIB address
  51    28 A6    3C  244F  6753                MOVZWL  SHR_W_QUOTA(R6),R1              ; Convert to longword
  20 A0    51    C0  2453  6754                ADDL    R1,JIB$L_BYTCNT(R0)             ; Return byte count quota
  24 A0    51    C0  2457  6755                ADDL    R1,JIB$L_BYTLM(R0)              ; ..and byte limit quota
00C8 C5    51    A2  245B  6756                SUBW    R1,UCB$W_XQ_QUOTA(R5)           ; Decrease the current quota
018B C5    51    A2  2460  6757                SUBW    R1,UCB$W_XQ_TOTQUO(R5)          ;  and the total quota
     50    56    D0  2465  6758 60$:           MOVL    R6,R0                           ; Copy SHR structure address
 00000000'GF    17  2468  6759                JMP     G^COM$DRVDEALMEM                ; Deallocate the structure
                    246E  6760
                    246E  6761 ;
                    246E  6762 ; Bug check on error
                    246E  6763 ;
                    246E  6764 90$:    BUG_CHECK NOBUFPCKT,FATAL
```

```
                     2472  6766          .SBTTL  CANCEL - CANCEL I/O ON UNIT
                     2472  6767 ;++
                     2472  6768 ; CANCEL - CANCEL I/O ON UNIT
                     2472  6769 ;
                     2472  6770 ; Functional description:
                     2472  6771 ;
                     2472  6772 ; This routine is used to cancel specific or all I/O pending on an XQ unit.
                     2472  6773 ;
                     2472  6774 ; Inputs:
                     2472  6775 ;
                     2472  6776 ;       R2 = Channel index number
                     2472  6777 ;       R4 = PCB address (or zero)
                     2472  6778 ;       R5 = UCB address
                     2472  6779 ;       R8 = Cancel reason code (CAN$C_DASSGN or CAN$C_CANCEL)
                     2472  6780 ;
                     2472  6781 ;
                     2472  6782 ;       IPL = FIPL
                     2472  6783 ;
                     2472  6784 ; Outputs:
                     2472  6785 ;
                     2472  6786 ;       R3-R5 are preserved.
                     2472  6787 ;       R0-R2 are destroyed.
                     2472  6788 ;
                     2472  6789 ;--
                     2472  6790
                     2472  6791 CANCEL::                                    ; Cancel I/O
00D8 8F    BB        2472  6792          PUSHR   #^M<R3,R4,R6,R7>           ; Save registers
      03   E1        2476  6793          BBC     #UCB$V_XQ_SHARE,-          ; Br if not a shared UCB
1A 68 A5             2478  6794                   UCB$W_DEVSTS(R5),2$       ;  perform regular $CANCEL
                     247B  6795 ;
                     247B  6796 ; Try to find SHR data structure
                     247B  6797 ;
013D   30            247B  6798          BSBW    FIND_SHR                   ; Check PID and CHAN
  15   12            247E  6799          BNEQ    2$                         ; Br if NO MATCH, maybe last $DASSGN
                     2480  6800 ;
                     2480  6801 ;       Match found - clear inited bit and clean up all I/O on SHR data
                     2480  6802 ;       structure.
                     2480  6803 ;
                     2480  6804 ;       We will Delete the SHR structure if this is a $DASSGN function
                     2480  6805 ;       request. We will get this function when called from SYS$DASSGN
                     2480  6806 ;       system service and so we will have to delete the SHR structure
                     2480  6807 ;       and decrement the reference count. Note that the reference count
                     2480  6808 ;       can never reach zero. Therefore, SYS$DASSGN will decrement the
                     2480  6809 ;       reference count on exit and we will be called again. This time
                     2480  6810 ;       there will be no match on the PID/CHAN and so the UCB will be
                     2480  6811 ;       cleaned up and deleted.
                     2480  6812 ;
   56   51   D0       2480  6813          MOVL    R1,R6                      ; Copy SHR address
54   24 A5   D0       2483  6814          MOVL    UCB$L_CRB(R5),R4           ; Get CRB address
54   10 A4   D0       2487  6815          MOVL    CRB$L_AUXSTRUC(R4),R4      ; Get CDB address
  FF08   30            248B  6816          BSBW    CLEANUP_SHR                ; Cleanup the SHR data structure
                     248E  6817
                     248E  6818          ASSUME  CAN$C_DASSGN EQ 1
   58   D7            248E  6819          DECL    R8                         ; Deassign request?
   37   12            2490  6820          BNEQ    10$                        ; Br if no - all done
  FF75   30            2492  6821          BSBW    DELETE_SHR                 ; Else, delete the SHR data stucture
                     2495  6822                                             ; And NOW perform like a NON-SHARED
```

```
                                2495  6823                                          ; unit.
                                2495  6824 ;
                                2495  6825 ; Non-shared unit - perform $CANCEL function.
                                2495  6826 ;
            5C A5    B5         2495  6827 2$:      TSTW    UCB$W_REFC(R5)          ; Last reference?
               34    12         2498  6828          BNEQ    20$                     ; Br if no - do selective cancel
         54    24 A5  D0         249A  6829 3$:      MOVL    UCB$L_CRB(R5),R4        ; Get CRB address
         54    10 A4  D0         249E  6830          MOVL    CRB$L_AUXSTRUC(R4),R4   ; Get CDB address
               FC65   30         24A2  6831          BSBW    SHUTDOWN                ; Shutdown entire unit
                  00  E0         24A5  6832          BBS     #CDB_STS_V_INITED,-     ; Br if QNA is still inited
            0A 024A C4           24A7  6833                  CDB_B_STS(R4),5$        ;
         024E C4    01  CE       24AB  6834          MNEGL   #1,CDB_G_PHA(R4)        ; Reset physical address
         0252 C4    01  AE       24B0  6835          MNEGW   #1,CDB_G_PHA+4(R4)      ;       ...
                                24B5  6836 ;
                                24B5  6837 ; When this is the last reference to the unit, reset the CPID of the UCB.
                                24B5  6838 ;
               10   A8           24B5  6839 5$:      BISW    S^#UCB$M_ONLINE,-       ; Set the UNIT to ONLINE
                  64 A5         24B7  6840                  UCB$W_STS(R5)           ;
               00BC C5  D5       24B9  6841          TSTL    UCB$L_XQ_CPID(R5)       ; Did we save the Creator PID?
                  0A  13         24BD  6842          BEQL    10$                     ; Br if not
         20 A5    00BC C5  D0    24BF  6843          MOVL    UCB$L_XQ_CPID(R5),UCB$L_CPID(R5) ; Else, restore Creator PID
               00BC C5  D4       24C5  6844          CLRL    UCB$L_XQ_CPID(R5)       ; Never again!!
               00D8 8F  BA       24C9  6845 10$:     POPR    #^M<R3,R4,R6,R7>        ; Restore registers
                        05       24CD  6846          RSB
                                24CE  6847 ;
                                24CE  6848 ; Abort all associated receive packets on UCB queue
                                24CE  6849 ;
                  04    E1       24CE  6850 20$:     BBC     #UCB$V_ONLINE,-         ; Br if not online
               F6 64 A5           24D0  6851                   UCB$W_STS(R5),10$      ;
                                24D3  6852
                                24D3  6853          ASSUME  UCB$V_XQ_INITED  EQ  0
            F2 68 A5    E9      24D3  6854          BLBC    UCB$W_DEVSTS(R5),10$    ; Br if not inited
         56    00A8 C5  9E       24D7  6855          MOVAB   UCB$Q_XQ_RCVREQ(R5),R6  ; Get address of receive queue
                  2D    10      24DC  6856          BSBB    CHECKER                 ; Check packets on queue
                                24DE  6857 ;
                                24DE  6858 ; Abort all xmit requests on CDB queue
                                24DE  6859 ;
            57    24 A5  D0       24DE  6860          MOVL    UCB$L_CRB(R5),R7        ; Get CRB address
            57    10 A7  D0       24E2  6861          MOVL    CRB$L_AUXSTRUC(R7),R7   ; Get CDB address
                  00    E1       24E6  6862          BBC     #CDB_STS_V_INITED,-     ; Br if not inited
            DD 024A C7           24E8  6863                  CDB_B_STS(R7),10$       ;
                                24EC  6864          DSBINT  UCB$B_DIPL(R5)          ; Sync access to CDB
         57    00DC C7  9E       24F3  6865          MOVAB   CDB_Q_QUEUES(R7),R7     ; Get start of queues
               58    01  9A      24F8  6866          MOVZBL  S^#CDB_C_ABORTS,R8      ; Get number of queues we can abort on
               56    57  D0      24FB  6867 30$:     MOVL    R7,R6                   ; Set address of next queue
                     48  10      24FE  6868          BSBB    CXB_CHECKER             ; Check CXBs on this queue
               57    08  C0      2500  6869          ADDL    #8,R7                   ; Skip to next queue
                  F5 58  F5      2503  6870          SOBGTR  R8,30$                  ; Loop thru queues
                                2506  6871          ENBINT                          ; Enable interrupts
                        BE  11   2509  6872          BRB     10$                     ; Exit from cancel
                                250B  6873 ;
                                250B  6874 ; Subroutine to scan queue for match on all packets
                                250B  6875 ;
                                250B  6876 CHECKER:
               53    66  D0      250B  6877          MOVL    (R6),R3                 ; Get next entry
               56    53  D1      250E  6878 10$:     CMPL    R3,R6                   ; End of list?
                     11  13      2511  6879          BEQL    30$                     ; Br if yes
```

```
                     10   10  2513  6880          BSBB    CHECKPKT                ; Cancel if appropriate match
                     08   12  2515  6881          BNEQ    20$                     ; Br if no match
                 53  63   0F  2517  6882          REMQUE  (R3),R3                 ; Remove from list
                   F6A5   30  251A  6883          BSBW    ABORT_PKT               ; Complete the I/O request
                     EC   11  251D  6884          BRB     CHECKER                 ; Look for more
                 53  63   D0  251F  6885 20$:     MOVL    (R3),R3                 ; Travel link
                     EA   11  2522  6886          BRB     10$                     ; Look for more
                          05  2524  6887 30$:     RSB                             ; Return to caller
                              2525  6888 ;
                              2525  6889 ; Subroutine to check for specific cancel
                              2525  6890 ;
                              2525  6891 CHECKPKT:
                  0C A3   D5  2525  6892          TSTL    IRP$L_PID(R3)           ; Is this an Internal IRP?
                     11   19  2528  6893          BLSS    30$                     ; Br if yes
                     06   12  252A  6894          BNEQ    10$                     ; Br if valid PID
                              252C  6895
                     54   D5  252C  6896          TSTL    R4                      ; Valid PCB?
                     17   12  252E  6897          BNEQ    50$                     ; Br if yes, no match
                     11   11  2530  6898          BRB     40$                     ; Else, test CHAN
                              2532  6899
              0C A3  60 A4   D1  2532  6900 10$:     CMPL    PCB$L_PID(R4),IRP$L_PID(R3) ; PID match?
                     0E   12  2537  6901          BNEQ    50$                     ; Br if no
                     08   11  2539  6902          BRB     40$                     ; Try CHAN match
                              253B  6903
          00B8 C5  60 A4   D1  253B  6904 30$:     CMPL    PCB$L_PID(R4),UCB$L_XQ_PID(R5) ; IS this the starter's PID?
                     04   12  2541  6905          BNEQ    50$                     ; Br if no
                28 A3  52   B1  2543  6906 40$:     CMPW    R2,IRP$W_CHAN(R3)       ; Channel match?
                     05   05  2547  6907 50$:     RSB                             ; Return to caller
                              2548  6908
                              2548  6909 CXB_CHECKER:
                  53  66   D0  2548  6910          MOVL    (R6),R3                 ; Get next entry
                  56  53   D1  254B  6911 10$:     CMPL    R3,R6                   ; End of list?
                     38   13  254E  6912          BEQL    30$                     ; Br if yes
                     37   10  2550  6913          BSBB    CXB_CHECKPKT            ; Cancel if appropriate match
                     2F   12  2552  6914          BNEQ    20$                     ; Br if no match
                  53  63   0F  2554  6915          REMQUE  (R3),R3                 ; Remove from CXB list
                              2557  6916          ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
                16 24 A3  E8  2557  6917          BLBS    CXB$L_T_IRP(R3),16$     ; Br if not IRP address -> FAST interface
               53  24 A3  D0  255B  6918          MOVL    CXB$L_T_IRP(R3),R3      ; Else, get IRP address
            50  0094 C3  D0  255F  6919          MOVL    IRP$L_XQ_SETUP(R3),R0   ; Get address of SETUP mode buffer
                     06   13  2564  6920          BEQL    13$                     ; Br if none
             00000000'GF  16  2566  6921          JSB     G^COM$DRVDEALMEM        ; Else, deallocate the buffer
                   F653   30  256C  6922 13$:     BSBW    ABORT_PKT               ; Complete the I/O request
                     D7   11  256F  6923          BRB     CXB_CHECKER             ; Look for more
                     54   DD  2571  6924 16$:     PUSHL   R4                      ; Save R4
            54   018D C5   D0  2573  6925          MOVL    UCB$L_XQ_FFI(R5),R4     ; Get FFI block address
                  50  2C   3C  2578  6926          MOVZWL  #SS$_ABORT,R0           ; Set status return
                14 B4   16  257B  6927          JSB     @FFI$L_XMIT_DONE(R4)    ; Complete the XMIT CXB
                     54 8ED0  257E  6928          POPL    R4                      ; Restore R4
                     C5   11  2581  6929          BRB     CXB_CHECKER             ; Look for more
                  53  63   D0  2583  6930 20$:     MOVL    (R3),R3                 ; Travel link
                     C3   11  2586  6931          BRB     10$                     ; Look for more
                          05  2588  6932 30$:     RSB                             ; Return to caller
                              2589  6933 ;
                              2589  6934 ; Subroutine to check for specific cancel
                              2589  6935 ;
                              2589  6936 CXB_CHECKPKT:
```

```
                50   24 A3   D0   2589  6937          MOVL    CXB$L_T_IRP(R3),R0      ; Get (presumed) IRP address
                                  258D  6938          ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
                     23 50   E8   258D  6939          BLBS    R0,80$                  ; Br if not an IRP address
                     0C A0   D5   2590  6940          TSTL    IRP$L_PID(R0)           ; Is this an Internal IRP?
                        11   19   2593  6941          BLSS    30$                     ; Br if yes
                        06   12   2595  6942          BNEQ    10$                     ; Br if valid PID
                                  2597  6943
                        54   D5   2597  6944          TSTL    R4                      ; Valid PCB?
                        17   12   2599  6945          BNEQ    50$                     ; Br if yes, no match
                        11   11   259B  6946          BRB     40$                     ; Else, test CHAN
                                  259D  6947
          0C A0         60 A4   D1   259D  6948 10$:   CMPL    PCB$L_PID(R4),IRP$L_PID(R0) ; PID match?
                        0E   12   25A2  6949          BNEQ    50$                     ; Br if no
                        08   11   25A4  6950          BRB     40$                     ; Try CHAN match
                                  25A6  6951
      00B8 C5           60 A4   D1   25A6  6952 30$:   CMPL    PCB$L_PID(R4),UCB$L_XQ_PID(R5) ; IS this the starter's PID?
                        04   12   25AC  6953          BNEQ    50$                     ; Br if no
               28 A0    52   B1   25AE  6954 40$:   CMPW    R2,IRP$W_CHAN(R0)       ; Channel match?
                        05   25B2  6955 50$:   RSB                                   ; Return to caller
                                  25B3  6956
                                  25B3  6957          ; No IRP with CXB - FFI user
                                  25B3  6958
                        54   D5   25B3  6959 80$:   TSTL    R4                      ; No PCB?
                        FB   13   25B5  6960          BEQL    50$                     ; Br if true - abort I/O
                     53 53   D0   25B7  6961          MOVL    R3,R3                   ; Else, return Z-BIT clear
                        05   25BA  6962          RSB                                 ; Return to caller
                                  25BB  6963
```

```
                        25BB  6965          .SBTTL   SUBROUTINES TO FIND SHR DATA STRUCTURE GIVEN PCB AND CHAN
                        25BB  6966 ;+
                        25BB  6967 ; Subroutine to find SHR data structure for user
                        25BB  6968 ;
                        25BB  6969 ; Inputs:
                        25BB  6970 ;       R2 = Channel number
                        25BB  6971 ;       R4 = PCB address (or zero)
                        25BB  6972 ;       R5 = UCB address
                        25BB  6973 ;
                        25BB  6974 ; Outputs:
                        25BB  6975 ;       R1 = Address if SHR data structure if match
                        25BB  6976 ;       R0 is destroyed.
                        25BB  6977 ;       Z-Bit set then match.
                        25BB  6978 ;       Z-Bit clear then no match.
                        25BB  6979 ;-
                        25BB  6980
                        25BB  6981 FIND_SHR:                                    ; Try to find shared user
51   00C4 C5    D0      25BB  6982          MOVL    UCB$L_XQ_DEFUSR(R5),R1     ; Get address of default user
             04 13      25C0  6983          BEQL    10$                        ; Br if no default user
             1C 10      25C2  6984          BSBB    90$                        ; Check for match
             19 13      25C4  6985          BEQL    40$                        ; Br if match
50   0098 C5    9E      25C6  6986 10$:     MOVAB   UCB$Q_XQ_SHARE(R5),R0      ; Save address of listhead
     51   50    D0      25CB  6987          MOVL    R0,R1                      ; Copy listhead address
                        25CE  6988          ASSUME  SHR_L_QFL EQ 0
     51   61    D0      25CE  6989 20$:     MOVL    (R1),R1                    ; Get next in list
     50   51    D1      25D1  6990          CMPL    R1,R0                      ; Back to start of list?
             06 13      25D4  6991          BEQL    30$                        ; Br if yes - no pid/chan match
             08 10      25D6  6992          BSBB    90$                        ; Check for match
             F4 12      25D8  6993          BNEQ    20$                        ; Br if none
             03 11      25DA  6994          BRB     40$                        ; Return in success
     50   50    D0      25DC  6995 30$:     MOVL    R0,R0                      ; Return match failure
             05 25DF    6996 40$:     RSB
                        25E0  6997
                        25E0  6998 ;+
                        25E0  6999 ; Subroutine to check if PID and SHR data base match up
                        25E0  7000 ;
                        25E0  7001 ; Inputs:
                        25E0  7002 ;       R1 = SHR address
                        25E0  7003 ;       R2 = Channel number
                        25E0  7004 ;       R4 = PCB address (or zero)
                        25E0  7005 ;
                        25E0  7006 ; Outputs:
                        25E0  7007 ;       Z-Bit set then match.
                        25E0  7008 ;       Z-Bit clear then no match.
                        25E0  7009 ;-
                        25E0  7010
                        25E0  7011 90$:                                         ; Check for match with SHR data base
          54    D5      25E0  7012          TSTL    R4                         ; Valid PCB address?
             07 12      25E2  7013          BNEQ    100$                       ; Br if yes
          0C A1    D5   25E4  7014          TSTL    SHR_L_PID(R1)              ; Zero PID?
             0D 12      25E7  7015          BNEQ    140$                       ; Br if not
             07 11      25E9  7016          BRB     110$                       ; Try for CHAN
0C A1     60 A4    D1   25EB  7017 100$:    CMPL    PCB$L_PID(R4),SHR_L_PID(R1) ; PIDs match?
             04 12      25F0  7018          BNEQ    140$                       ; Br if no - try for next
   10 A1     52    B1   25F2  7019 110$:    CMPW    R2,SHR_W_CHAN(R1)          ; Channels match?
             05 25F6    7020 140$:    RSB                                     ; Return to caller
```

```
                        25F7  7022          .SBTTL  FIND_POINT_UCB  - Find the point to point UCB
                        25F7  7023 ;++
                        25F7  7024 ; FIND_POINT_UCB - Find the point-to-point UCB
                        25F7  7025 ;
                        25F7  7026 ; Functional description:
                        25F7  7027 ;
                        25F7  7028 ; This routine is called to find the point-to-point UCB for some received
                        25F7  7029 ; message. This is only needed when the protocol is in the startup state.
                        25F7  7030 ;
                        25F7  7031 ; Inputs:
                        25F7  7032 ;
                        25F7  7033 ;       R1 = Protocol type (startup)
                        25F7  7034 ;       R2 = MSG buffer address
                        25F7  7035 ;       R4 = CDB address
                        25F7  7036 ;
                        25F7  7037 ;
                        25F7  7038 ;       IPL = FIPL
                        25F7  7039 ;
                        25F7  7040 ; Outputs:
                        25F7  7041 ;
                        25F7  7042 ;       R0 = Status return for request
                        25F7  7043 ;       All other registers are preserved
                        25F7  7044 ;--
                        25F7  7045
                        25F7  7046 FIND_POINT_UCB:                              ;% Find the point-to-point UCB
    006E 8F     BB      25F7  7047          PUSHR   #^M<R1,R2,R3,R5,R6>         ;% Save registers
         50     D4      25FB  7048          CLRL    R0                          ;% Assume failure
   46 A2  01     B1      25FD  7049          CMPW    #1,CXB$W_R_SIZE(R2)         ;% Is buffer the right size?
         54     12      2601  7050          BNEQ    99$                         ;% Br if not!
48 A2    AA     8F      91    2603  7051          CMPB    #^XAA,CXB$W_R_SIZE+2(R2)    ;% Is first byte start byte?
         07     13      2608  7052          BEQL    10$                         ;% Br if yes
48 A2    AB     8F      91    260A  7053          CMPB    #^XAB,CXB$W_R_SIZE+2(R2)    ;% Is first byte stack byte?
         73     12      260F  7054          BNEQ    90$                         ;% Br if not
 55  0118 C4     D0      2611  7055 10$:     MOVL    CDB_L_UCB0(R4),R5           ;% Get UCB address of unit 0
   55  30 A5     D0      2616  7056 20$:     MOVL    UCB$L_LINK(R5),R5           ;% Get address of next UCB
         68     13      261A  7057          BEQL    90$                         ;% Br if end of list
                        261C  7058          ASSUME  UCB$V_XQ_INITED EQ 0
      F6 68 A5  E9      261C  7059          BLBC    UCB$W_DEVSTS(R5),20$        ;% Br if not inited
         00     91      2620  7060          CMPB    #NMA$C_LINPR_POI,-          ;% Is this a point-to-point user?
      00D8 C5             2622  7061                  UCB$B_XQ_PRO(R5)            ;%
         EF     12      2625  7062          BNEQ    20$                         ;% Br if not
      00CC C5     D1      2627  7063          CMPL    UCB$G_XQ_DES(R5),-          ;% Does the destination match?
        3E A2             262B  7064                  CXB$G_R_SRC(R2)             ;%
         E7     12      262D  7065          BNEQ    20$                         ;% Br if not
      00D0 C5     B1      262F  7066          CMPW    UCB$G_XQ_DES+4(R5),-        ;% Still match?
        42 A2             2633  7067                  CXB$G_R_SRC+4(R2)           ;%
         DF     12      2635  7068          BNEQ    20$                         ;% Br if not
48 A2    AB     8F      91    2637  7069          CMPB    #^XAB,CXB$W_R_SIZE+2(R2)    ;% Is first byte stack byte?
         14     12      263C  7070          BNEQ    40$                         ;% Br if no
05 68 A5     05     E4      263E  7071          BBSC    #UCB$V_XQ_START,UCB$W_DEVSTS(R5),30$ ;% Clear Start state
28 68 A5     06     E5      2643  7072          BBCC    #UCB$V_XQ_STACK,UCB$W_DEVSTS(R5),60$ ;% We were in RUN, ignore
 52  0191 C5     D0      2648  7073 30$:     MOVL    UCB$L_XQ_STIRP(R5),R2       ;% Get start IRP
          FCF2   30      264D  7074          BSBW    BLD_STRT_IRP                ;% Send stack!
         1E     11      2650  7075          BRB     60$                         ;% Then send data
48 A2    AA     8F      91    2652  7076 40$:     CMPB    #^XAA,CXB$W_R_SIZE+2(R2)    ;% Is first byte start byte?
         2B     12      2657  7077 99$:     BNEQ    90$                         ;% Br if not, ignore it
      0060 8F     B3      2659  7078          BITW    #UCB$M_XQ_START!UCB$M_XQ_STACK,- ;% Are we in startup states?
```

```
                68 A5          265D  7079                    UCB$W_DEVSTS(R5)
                   28   13     265F  7080            BEQL    120$                    ;% Br if not, start recvd in RUN
                                2661  7081            CLRBIT  #UCB$V_XQ_START,UCB$W_DEVSTS(R5);% Clear starting bit
52            0191 C5    D0     2666  7082            MOVL    UCB$L_XQ_STIRP(R5),R2   ;% Get start IRP
                  FCD4    30     266B  7083            BSBW    BLD_STRT_IRP            ;% Send stack!
                   11     11     266E  7084            BRB     80$                     ;% Wait for stack
53            00B0 D5    0F     2670  7085 60$:        REMQUE  @UCB$Q_XQ_XMTREQ(R5),R3 ;% Get transmit IRPs
                   07     1D     2675  7086            BVS     70$                     ;% Br if no more
00E0 D4           63      0E     2677  7087            INSQUE  (R3),@CDB_Q_XMTREQ+4(R4);% Insert IRPs onto xmit queue
                   F2     11     267C  7088            BRB     60$                     ;% Look for more
                DE12      30     267E  7089 70$:        BSBW    XMT_ALT_START           ;% Startup the xmit process
                50  01    9A     2681  7090 80$:        MOVZBL  #1,R0                   ;% Return success
                                2684  7091
                006E 8F    BA     2684  7092 90$:        POPR    #^M<R1,R2,R3,R5,R6>     ;% Restore registers
                           05     2688  7093            RSB                             ;% Return to caller
                                2689  7094
                                2689  7095 ;
                                2689  7096 ; Start received in run mode
                                2689  7097 ;
                0098 C5    D1     2689  7098 120$:       CMPL    UCB$Q_XQ_SHARE(R5),-    ;% Is limited queue empty?
                0098 D5           268D  7099                    @UCB$Q_XQ_SHARE(R5)     ;%
                   F2   13     2690  7100            BEQL    90$                     ;% Br if yes, no IRPs
                                2692  7101            CLRBIT  #XM$V_STS_ACTIVE,UCB$L_DEVDEPEND(R5) ;% Clear active bit
                                2697  7102            SETBIT  #XM$V_ERR_START,UCB$L_DEVDEPEND(R5) ;% Indicate cause of error
                0060 8F    AA     269C  7103            BICW    #UCB$M_XQ_START!UCB$M_XQ_STACK,-;% Clear start and stack flags
                  68 A5           26A0  7104                    UCB$W_DEVSTS(R5)        ;%
                                26A2  7105            CLRBIT  #UCB$V_XQ_RUN,UCB$W_DEVSTS(R5) ;% Clear the RUN flag
56            0098 C5    D0     26A7  7106            MOVL    UCB$Q_XQ_SHARE(R5),R6   ;% Get address of share structure
                  FCE7    30     26AC  7107            BSBW    CLEANUP_SHR             ;% Cleanup all pending I/O
                   D3     11     26AF  7108            BRB     90$                     ;% Exit
                                26B1  7109
```

```
                                26B1  7111          .SBTTL  ADD_MULTI - ADD UP ALL THE MULTICAST ADDRESSES
                                26B1  7112 ;++
                                26B1  7113 ; ADD_MULTI - ADD UP ALL THE MULTICAST ADDRESSES
                                26B1  7114 ;
                                26B1  7115 ; Functional description:
                                26B1  7116 ;
                                26B1  7117 ; This routine is called to combine all the per protocol type multicast
                                26B1  7118 ; addresses into a single list in the CDB. If the sum of all multicast
                                26B1  7119 ; addresses is greater than the QNA can manage, then an error is returned.
                                26B1  7120 ;
                                26B1  7121 ; Inputs:
                                26B1  7122 ;
                                26B1  7123 ;       R5 = UCB address
                                26B1  7124 ;
                                26B1  7125 ;
                                26B1  7126 ;       IPL = FIPL
                                26B1  7127 ;
                                26B1  7128 ; Outputs:
                                26B1  7129 ;
                                26B1  7130 ;       R0 = Status return for request
                                26B1  7131 ;       R1,R2 are destroyed
                                26B1  7132 ;       R3-R5 are preserved
                                26B1  7133 ;
                                26B1  7134 ; Implicit outputs:
                                26B1  7135 ;
                                26B1  7136 ;       CDB_B_MLTTBL = Number of multicast addresses in CDB_G_MLTTBL
                                26B1  7137 ;       CDB_G_MLTTBL = New multicast address list
                                26B1  7138 ;--
                                26B1  7139
                                26B1  7140 ADD_MULTI:                              ; Add up all the multicast addresses
          00D8  8F      BB      26B1  7141          PUSHR   #^M<R3,R4,R6,R7>      ; Save registers
     54     24  A5      D0      26B5  7142          MOVL    UCB$L_CRB(R5),R4      ; Get CRB address
     54     10  A4      D0      26B9  7143          MOVL    CRB$L_AUXSTRUC(R4),R4 ; Get CDB address
          0261  C4      94      26BD  7144          CLRB    CDB_B_MLTTBL(R4)      ; Reset number of entries
                                26C1  7145          PUSHQ   R4                    ; Save CDB and UCB addresses
00   02AA  C4    00      2C      26C4  7146          MOVC5   #0,CDB_G_MLTTBL(R4),#0,- ; Zero the structure
02AA C4    0048  8F              26CA  7147                  #6*MAX_C_MLT,CDB_G_MLTTBL(R4) ;
                                26D0  7148          POPQ    R4                    ; Restore CDB and UCB addresses
          53    0D      9A      26D3  7149          MOVZBL  #MAX_C_MLT+1,R3       ; Error if 1 more multicast address
                                26D6  7150                                        ;  than we can handle
          50    01      CE      26D6  7151          MNEGL   #1,R0                 ; Assume success
     56   02AA  C4      9E      26D9  7152          MOVAB   CDB_G_MLTTBL(R4),R6   ; Get address of Multicast table
     57     28  A5      D0      26DE  7153          MOVL    UCB$L_DDB(R5),R7      ; Get DDB address
     57     04  A7      D0      26E2  7154          MOVL    DDB$L_UCB(R7),R7      ; Get 1st UCB address
     57     30  A7      D0      26E6  7155 10$:     MOVL    UCB$L_LINK(R7),R7     ; Get next UCB in list
                37      13      26EA  7156          BEQL    50$                   ; Br if no more UCB's
                                26EC  7157
                                26EC  7158          ASSUME  UCB$V_XQ_INITED  EQ  0
        F6  68  A5      E9      26EC  7159          BLBC    UCB$W_DEVSTS(R5),10$  ; Br if not inited
     52   00E7  C7      9E      26F0  7160          MOVAB   UCB$G_XQ_MULTI(R7),R2 ; Get address of Multicast list
     51   00E5  C7      9A      26F5  7161          MOVZBL  UCB$B_XQ_MULTI(R7),R1 ; Set number addresses for UCB
                62      D5      26FA  7162 20$:     TSTL    (R2)                  ; Is this field unused?
                05      12      26FC  7163          BNEQ    25$                   ; Br if no
             04 A2      B5      26FE  7164          TSTW    4(R2)                 ; Really?
                16      13      2701  7165          BEQL    30$                   ; Yes - skip it
                53      97      2703  7166 25$:     DECB    R3                    ; One less available slot in CDB
                1A      13      2705  7167          BEQL    40$                   ; Br if none left - error
```

```
                86    62    D0  2707  7168          MOVL    (R2),(R6)+              ; Else, insert next address
          86      04  A2    B0  270A  7169          MOVW    4(R2),(R6)+             ;       ...
              0261    C4    96  270E  7170          INCB    CDB_B_MLTTBL(R4)        ; Count one more in list
              0261    C4    91  2712  7171          CMPB    CDB_B_MLTTBL(R4),-      ; Is there enough room?
                      0C        2716  7172           #MAX_C_MLT                    ;       ...
                      08    1A  2717  7173          BGTRU   40$                     ; Br if no - error
                52    06    C0  2719  7174  30$:     ADDL    #6,R2                   ; Skip to next entry
                  DB  51    F5  271C  7175          SOBGTR  R1,20$                  ; Br if more
                      C5    11  271F  7176          BRB     10$                     ; Else, skip to next UCB
                            2721  7177
                      50    D4  2721  7178  40$:     CLRL    R0                      ; Return failure
                00D8  8F    BA  2723  7179  50$:     POPR    #^M<R3,R4,R6,R7>        ; Restore registers
                            05  2727  7180          RSB
```

```
                                2728  7182          .SBTTL  MOVE_MULTI - COPY THE MULTICAST ADDRESS LIST
                                2728  7183 ;++
                                2728  7184 ; MOVE_MULTI - COPY THE MULTICAST ADDRESS LIST
                                2728  7185 ;
                                2728  7186 ; Functional description:
                                2728  7187 ;
                                2728  7188 ; This routine is called to copy the multicast address list from the
                                2728  7189 ; generation table to the actual list.
                                2728  7190 ;
                                2728  7191 ; Inputs:
                                2728  7192 ;
                                2728  7193 ;       R4 = CDB address
                                2728  7194 ;
                                2728  7195 ; Outputs:
                                2728  7196 ;
                                2728  7197 ;       All registers are preserved.
                                2728  7198 ;
                                2728  7199 ; Implicit outputs:
                                2728  7200 ;
                                2728  7201 ;       CDB_B_MULTI = Number of multicast addresses in CDB_G_MULTI
                                2728  7202 ;       CDB_G_MULTI = New multicast address list
                                2728  7203 ;--
                                2728  7204
                                2728  7205 MOVE_MULTI:                                    ; Move the multicast address list
                             3F BB 2728  7206          PUSHR   #^M<R0,R1,R2,R3,R4,R5>   ; Save registers
0262 C4  02AA C4  0048 8F    28 272A  7207          MOVC3   #MAX_C_MLT*6,CDB_G_MLTTBL(R4),CDB_G_MULTI(R4) ; Copy list
                             3F BA 2734  7208          POPR    #^M<R0,R1,R2,R3,R4,R5>   ; Restore registers
         0260 C4  0261 C4    90 2736  7209          MOVB    CDB_B_MLTTBL(R4),CDB_B_MULTI(R4) ; Set number of entries
                             05 273D  7210          RSB
```

```
                                                J 2
                     273E  7212          .SBTTL  ROUTINES TO SAVE/RESTORE UCB'S MULTICAST ADDRESS LIST
                     273E  7213 ;++
                     273E  7214 ; ROUTINES TO SAVE/RESTORE UCB'S MULTICAST ADDRESS LIST
                     273E  7215 ;
                     273E  7216 ; Functional description:
                     273E  7217 ;
                     273E  7218 ; These routines are called to save or restore the multicast address list
                     273E  7219 ; in the UCB.
                     273E  7220 ;
                     273E  7221 ; Inputs:
                     273E  7222 ;
                     273E  7223 ;       R5 = UCB address
                     273E  7224 ;
                     273E  7225 ; Outputs:
                     273E  7226 ;
                     273E  7227 ;       All registers are preserved.
                     273E  7228 ;--
                     273E  7229
                     273E  7230 SAV_MULTI:                                      ; Save multicast address list
                3F   BB   273E  7231          PUSHR   #^M<R0,R1,R2,R3,R4,R5>  ; Save registers
00E7 C5    0048 8F   28   2740  7232          MOVC3   #6*MAX_C_MLT,UCB$G_XQ_MULTI(R5),- ; Save multicast addresses
           012F C5        2747  7233                  UCB$G_XQ_MLTTBL(R5)       ;       ...
                3F   BA   274A  7234          POPR    #^M<R0,R1,R2,R3,R4,R5>  ; Restore registers
           00E5 C5   90   274C  7235          MOVB    UCB$B_XQ_MULTI(R5),-    ; Save count of multicast addresses
           00E6 C5        2750  7236                  UCB$B_XQ_MLTTBL(R5)       ;       ...
                05   RSB   2753  7237          RSB                             ; Return to caller
                     2754  7238
                     2754  7239 RES_MULTI:                                    ; Restore multicast address list
                3F   BB   2754  7240          PUSHR   #^M<R0,R1,R2,R3,R4,R5>  ; Save registers
012F C5    0048 8F   28   2756  7241          MOVC3   #6*MAX_C_MLT,UCB$G_XQ_MLTTBL(R5),- ; Restore multicast addresses
           00E7 C5        275D  7242                  UCB$G_XQ_MULTI(R5)        ;       ...
                3F   BA   2760  7243          POPR    #^M<R0,R1,R2,R3,R4,R5>  ; Restore registers
           00E6 C5   90   2762  7244          MOVB    UCB$B_XQ_MLTTBL(R5),-   ; Restore count of multicast addresses
           00E5 C5        2766  7245                  UCB$B_XQ_MULTI(R5)        ;       ...
                05   RSB   2769  7246          RSB                             ; Return to caller
```

```
                     276A  7248          .SBTTL  VALIDATE_P2 - VALIDATE P2 BUFFER PARAMETERS
                     276A  7249
                     276A  7250 ;++
                     276A  7251 ; VALIDATE_P2 - Validate P2 buffer parameters
                     276A  7252 ;
                     276A  7253 ; This routine is called to validate the P2 buffer parameters. The parameters
                     276A  7254 ; are checked against a parameter table which verifies that the minimum value
                     276A  7255 ; and maximum value is not violated, and that invalid status flags are not set.
                     276A  7256 ; If the parameter is a string, then the string must not exceed the maximum
                     276A  7257 ; string count for this parameter.
                     276A  7258 ;
                     276A  7259 ; Inputs:
                     276A  7260 ;
                     276A  7261 ;       R2 = Address of verification table
                     276A  7262 ;       R3 = IRP address
                     276A  7263 ;       R5 = UCB address
                     276A  7264 ;
                     276A  7265 ;
                     276A  7266 ; Outputs:
                     276A  7267 ;
                     276A  7268 ;       R0 = Status return for request
                     276A  7269 ;
                     276A  7270 ; If no error:
                     276A  7271 ;       R1 = Address of parameter verification table
                     276A  7272 ; If error:
                     276A  7273 ;       R1 = Bad parameter value
                     276A  7274 ;
                     276A  7275 ;       All other registers are preserved.
                     276A  7276 ;
                     276A  7277 ;--
                     276A  7278
                     276A  7279 VALIDATE_P2::                                ; Validate P2 buffer parameters
    03DC 8F     BB   276A  7280          PUSHR   #^M<R2,R3,R4,R6,R7,R8,R9> ; Save registers
56    2C A3     D0   276E  7281          MOVL    IRP$L_SVAPTE(R3),R6       ; Get system P2 buffer address
      03        12   2772  7282          BNEQ    20$                       ; Br if a system buffer
       0124     31   2774  7283 10$:     BRW     150$                      ; Else, leave
                     2777  7284
54    24 A5     D0   2777  7285 20$:     MOVL    UCB$L_CRB(R5),R4          ; Get CRB address
54    10 A4     D0   277B  7286          MOVL    CRB$L_AUXSTRUC(R4),R4     ; Get CDB address
   56    66     D0   277F  7287          MOVL    P2B_L_POINTER(R6),R6      ; Point to start of P2 data
58    32 A3     3C   2782  7288          MOVZWL  IRP$W_BCNT(R3),R8         ; Get size of P2 buffer
                     2786  7289 ;
                     2786  7290 ; Loop to check next parameter in P2 buffer
                     2786  7291 ;
   58    02     C2   2786  7292 30$:     SUBL    #2,R8                     ; Can we get another parameter?
         E9     19   2789  7293          BLSS    10$                       ; Br if no - all done
   51    86     3C   278B  7294          MOVZWL  (R6)+,R1                  ; Get parameter type from P2
                     278E  7295 ;
                     278E  7296 ; *** NOTE - R2 MUST be the very first item on the top of the stack
                     278E  7297 ;
   57    6E     D0   278E  7298          MOVL    (SP),R7                   ; Get verification table address
                     2791  7299 ;
                     2791  7300 ; Loop to check P2 buffer parameter to Line parameter table
                     2791  7301 ;
                     2791  7302          ASSUME  PRM_W_TYPE EQ 0
   50    87     B0   2791  7303 40$:     MOVW    (R7)+,R0                  ; Get parameter type code
         03     12   2794  7304          BNEQ    45$                       ; Br if NOT end of verify table
```

```
                0107   31   2796  7305 43$:    BRW     170$                        ; Else exit in error
                              2799  7306
                              2799  7307 45$:    ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
          59    87     9A   2799  7308         MOVZBL  (R7)+,R9                    ; Get flags byte
    50    F000  8F     AA   279C  7309         BICW    #^C<PRM_TYP_M_CODE>,R0      ; Clear all but type code
          50    51     B1   27A1  7310         CMPW    R1,R0                       ; Parameters match?
                17     13   27A4  7311         BEQL    50$                         ; Br if yes
          57    02     C0   27A6  7312         ADDL    #2,R7                       ; Skip offset word
                              27A9  7313         SKIP    PRM_FLG_V_MIN,R9,R7         ; Skip minimum value
                              27AF  7314         SKIP    PRM_FLG_V_MAX,R9,R7         ; Skip maximum value
                              27B5  7315         SKIP    PRM_FLG_V_INVALID,R9,R7     ; Skip invalid flags
                D4    11     27BB  7316         BRB     40$                         ; Try next parameter
                              27BD  7317 ;
                              27BD  7318 ; Match found - nullify if same value & check min,max,valid,invalid
                              27BD  7319 ;
          50    87     B0   27BD  7320 50$:    MOVW    (R7)+,R0                    ; Get offset + width
                0A     EF   27C0  7321         EXTZV   #PRM_OFF_V_WIDTH,-          ; Get width only
    52    50    06           27C2  7322                 #PRM_OFF_S_WIDTH,R0,R2      ;    ...
                00     EF   27C5  7323         EXTZV   #PRM_OFF_V_VALUE,-          ; Get offset only
    50    50    0A           27C7  7324                 #PRM_OFF_S_VALUE,R0,R0      ;    ...
 05       59    03     E0   27CA  7325         BBS     #PRM_FLG_V_CDB,R9,55$       ; Br if CDB datum
          50    55     C0   27CE  7326         ADDL    R5,R0                       ; Compute offset in UCB
                03     11   27D1  7327         BRB     57$                         ; Continue
          50    54     C0   27D3  7328 55$:    ADDL    R4,R0                       ; Compute offset in CDB
                              27D6  7329 57$:    ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
FB        A7    0C     E0   27D6  7330         BBS     #PRM_TYP_V_STRING,-5(R7),95$ ; Br if string parameter
          58    04     C2   27DB  7331         SUBL    #4,R8                       ; Must be longword value
                B6     19   27DE  7332         BLSS    43$                         ; Br if error
          53    86     D0   27E0  7333         MOVL    (R6)+,R3                    ; Get parameter value
                              27E3  7334         CASE    R2,TYPE=B,LIMIT=#1,<-       ; Br to handler
                              27E3  7335                 60$,-                       ;   Byte value
                              27E3  7336                 70$,-                       ;   Word value
                              27E3  7337                 80$>                        ;   Longword value
                              27ED  7338 ;
                              27ED  7339 ; Byte value in structure
                              27ED  7340 ;
          60    53     91   27ED  7341 60$:    CMPB    R3,(R0)                     ; Is this the same?
                08     11   27F0  7342         BRB     90$                         ; Check result
                              27F2  7343 ;
                              27F2  7344 ; Word value
                              27F2  7345 ;
          60    53     B1   27F2  7346 70$:    CMPW    R3,(R0)                     ; Is this the same?
                03     11   27F5  7347         BRB     90$                         ; Check result
                              27F7  7348 ;
                              27F7  7349 ; Longword value
                              27F7  7350 ;
          60    53     D1   27F7  7351 80$:    CMPL    R3,(R0)                     ; Is this the same?
                6E     12   27FA  7352 90$:    BNEQ    100$                        ; Br if no - continue checks
0B0E      8F    51     B1   27FC  7353         CMPW    R1,#NMA$C_PCLI_PTY          ; Is this the protocol type?
                03     13   2801  7354         BEQL    91$                         ; Br if yes - always store this
             FA   A6     B4   2803  7355         CLRW    -6(R6)                      ; Nullify the parameter code
              008F       31   2806  7356 91$:    BRW     140$                        ; Try next parameter - skip checks
              0094       31   2809  7357 93$:    BRW     170$                        ; LONNGGG Branch to 170$
                              280C  7358 ;
                              280C  7359 ; String value
                              280C  7360 ;
          58    02     C2   280C  7361 95$:    SUBL    #2,R8                       ; Can we fetch string length?
```

```
                F8   19   280F  7362           BLSS    93$                     ; Br if no - error
          53    86   3C   2811  7363           MOVZWL  (R6)+,R3                ; Get string length
          58    53   C2   2814  7364           SUBL    R3,R8                   ; Is there room for string?
                F0   19   2817  7365           BLSS    93$                     ; Br if no - error
          52    53   B1   2819  7366           CMPW    R3,R2                   ; Is the string too long?
                EB   1A   281C  7367           BGTRU   93$                     ; Br if yes - error
          56    53   C0   281E  7368           ADDL    R3,R6                   ; Skip past string
     51   0B21  8F   B1   2821  7369           CMPW    #NMA$C_PCLI_DES,R1      ; Is this the destination address?
                07   13   2826  7370           BEQL    96$                     ; Br if yes
     51   0B04  8F   B1   2828  7371           CMPW    #NMA$C_PCLI_PHA,R1      ; Is this the physical address?
                08   12   282D  7372           BNEQ    97$                     ; Br if not
                  0270   30   282F  7373 96$:   BSBW    VALID_PHYAD             ; Validate the physical address
                6B 50    E9   2832  7374           BLBC    R0,170$                 ; Br if error in physical address
                61   11   2835  7375           BRB     140$                    ; Else, continue checking
     51   0B0F  8F   B1   2837  7376 97$:   CMPW    #NMA$C_PCLI_MCA,R1      ; Is this the multicast address list?
                3E   12   283C  7377           BNEQ    130$                    ; Br if no - okay
                  0225   30   283E  7378           BSBW    VALID_MULTI             ; Validate the multicast address list
                5C 50    E9   2841  7379           BLBC    R0,170$                 ; Br if error
                              2844  7380           DSBINT  UCB$B_FIPL(R5)          ; Sync access to UCB
                  FEF0   30   284B  7381           BSBW    SAV_MULTI               ; Save the multicast addresses
          0240  8F   BB   284E  7382           PUSHR   #^M<R6,R9>              ; Save registers
          56    53   C2   2852  7383           SUBL    R3,R6                   ; Backup pointer to start of list
          59    53   D0   2855  7384           MOVL    R3,R9                   ; Setup string count in R9
                  027F   30   2858  7385           BSBW    SET_MULTI               ; See if we can set new addresses
                              285B  7386                                           ; R0 = return status
          0240  8F   BA   285B  7387           POPR    #^M<R6,R9>              ; Restore registers
                  FEF2   30   285F  7388           BSBW    RES_MULTI               ; Restore the multicast list
                              2862  7389           ENBINT                          ; Restore IPL
                38 50    E9   2865  7390           BLBC    R0,170$                 ; Br if error
                12   11   2868  7391           BRB     130$                    ; Check if state okay
                              286A  7392
     05   59    00   E1   286A  7393 100$:  BBC     #PRM_FLG_V_MIN,R9,110$  ; Br if no minimum value
          87    53   B1   286E  7394           CMPW    R3,(R7)+                ; Is the value too small?
                2D   1F   2871  7395           BLSSU   170$                    ; Br if yes - error
     05   59    01   E1   2873  7396 110$:  BBC     #PRM_FLG_V_MAX,R9,130$  ; Br if no maximum value
          87    53   B1   2877  7397           CMPW    R3,(R7)+                ; Is the value too big?
                24   1A   287A  7398           BGTRU   170$                    ; Br if yes - error
     18   59    02   E1   287C  7399 130$:  BBC     #PRM_FLG_V_INVALID,R9,140$ ; Br if no invalid flags
          52    87   B0   2880  7400           MOVW    (R7)+,R2                ; Get invalid flags
     06   59    03   E0   2883  7401           BBS     #PRM_FLG_V_CDB,R9,135$  ; Br if CDB datum
     68   A5    52   B3   2887  7402           BITW    R2,UCB$W_DEVSTS(R5)     ; Check UCB invalid bits
                09   11   288B  7403           BRB     137$                    ; Continue
                54   D5   288D  7404 135$:  TSTL    R4                      ; Is CDB present?
                07   13   288F  7405           BEQL    140$                    ; Br if no - okay
024A C4    52   93   2891  7406           BITB    R2,CDB_B_STS(R4)        ; Check CDB invalid bits
                08   12   2896  7407 137$:  BNEQ    170$                    ; Br on error
              FEEB   31   2898  7408 140$:  BRW     30$                     ; Loop if more parameters
                              289B  7409
          50    01   9A   289B  7410 150$:  MOVZBL  S^#SS$_NORMAL,R0          ; Set success return
                03   11   289E  7411           BRB     180$                    ; And return
                              28A0  7412
          50    14   9A   28A0  7413 170$:  MOVZBL  S^#SS$_BADPARAM,R0        ; Set error return
          03DC  8F   BA   28A3  7414 180$:  POPR    #^M<R2,R3,R4,R6,R7,R8,R9> ; Restore registers
                05   28A7  7415           RSB                             ; Return to caller
```

```
                     28A8  7417          .SBTTL  CHANGE_PARAM - UPDATE UCB/CDB BASED ON P2 BUFFER PARAMETERS
                     28A8  7418
                     28A8  7419 ;++
                     28A8  7420 ; CHANGE_PARAM - Update UCB/CDB with P2 buffer parameters
                     28A8  7421 ;
                     28A8  7422 ; This routine is called to update the UCB/CDB with the P2 buffer parameters.
                     28A8  7423 ; The parameters are stored in the appropriate cells of the UCB/CDB.
                     28A8  7424 ; This routine can only modify the LINE PARAMETERS.
                     28A8  7425 ;
                     28A8  7426 ; Inputs:
                     28A8  7427 ;
                     28A8  7428 ;       R2 = Address of verification table
                     28A8  7429 ;       R3 = IRP address
                     28A8  7430 ;       R5 = UCB address
                     28A8  7431 ;
                     28A8  7432 ;       IPL = FIPL
                     28A8  7433 ;
                     28A8  7434 ; Outputs:
                     28A8  7435 ;
                     28A8  7436 ;       R0 = destroyed.
                     28A8  7437 ;       All other registers are preserved.
                     28A8  7438 ;
                     28A8  7439 ;--
                     28A8  7440
                     28A8  7441 CHANGE_PARAM:                                ; Change the UCB/CDB parameters
    07DE 8F     BB   28A8  7442          PUSHR   #^M<R1,R2,R3,R4,R6,R7,R8,R9,R10> ; Save registers
    5A   52     D0   28AC  7443          MOVL    R2,R10                      ; Save table address
56  2C A3       D0   28AF  7444          MOVL    IRP$L_SVAPTE(R3),R6         ; Get system P2 buffer address
        03      12   28B3  7445          BNEQ    5$                          ; Br if system buffer
      00DB      31   28B5  7446 3$:      BRW     120$                        ; Else, return
                     28B8  7447
54  24 A5       D0   28B8  7448 5$:      MOVL    UCB$L_CRB(R5),R4            ; Get CRB address
54  10 A4       D0   28BC  7449          MOVL    CRB$L_AUXSTRUC(R4),R4       ; Get CDB address
   56   66      D0   28C0  7450          MOVL    P2B_L_POINTER(R6),R6        ; Point to start of data
58  32 A3       3C   28C3  7451          MOVZWL  IRP$W_BCNT(R3),R8           ; Get size of P2 buffer
                     28C7  7452 ;
                     28C7  7453 ; Loop to get next parameter from P2 buffer
                     28C7  7454 ;
    58   02     C2   28C7  7455 10$:     SUBL    #2,R8                       ; Try to get next parameter
         E9     19   28CA  7456          BLSS    3$                          ; Br if not there
    50   86     3C   28CC  7457          MOVZWL  (R6)+,R0                    ; Get parameter type from P2
         79     13   28CF  7458          BEQL    90$                         ; Br if null value parameter
    57   5A     D0   28D1  7459          MOVL    R10,R7                      ; Get verification table address
0B0E 8F   50     B1   28D4  7460          CMPW    R0,#NMA$C_PCLI_PTY          ; Is this the protocol type?
         04     12   28D9  7461          BNEQ    20$                         ; Br if not
         04     A8   28DB  7462          BISW    #UCB$M_XQ_PROTYP,-          ; Indicate that protocol type specified
      68 A5          28DD  7463                  UCB$W_DEVSTS(R5)            ;
                     28DF  7464 ;
                     28DF  7465 ; Loop to store buffer parameter in UCB/CDB
                     28DF  7466 ;
                     28DF  7467          ASSUME  PRM_W_TYPE EQ 0
    51   87     3C   28DF  7468 20$:     MOVZWL  (R7)+,R1                    ; Get parameter type code
         66     13   28E2  7469          BEQL    90$                         ; Br if end of verify table
51  F000 8F     AA   28E4  7470          BICW    #^C<PRM_TYP_M_CODE>,R1      ; Clear all but type code
                     28E9  7471          ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
    59   87     9A   28E9  7472          MOVZBL  (R7)+,R9                    ; Get flags byte
    51   50     B1   28EC  7473          CMPW    R0,R1                       ; Parameters match?
```

```
                17   13   28EF  7474          BEQL    30$                             ; Br if yes
            57  02   C0   28F1  7475          ADDL    #2,R7                           ; Skip offset word
                         28F4  7476          SKIP    PRM_FLG_V_MIN,R9,R7             ; Skip minimum value
                         28FA  7477          SKIP    PRM_FLG_V_MAX,R9,R7             ; Skip maximum value
                         2900  7478          SKIP    PRM_FLG_V_INVALID,R9,R7         ; Skip invalid flags
                    D7   11   2906  7479          BRB     20$                             ; Try next parameter
                         2908  7480 ;
                         2908  7481 ; Match found - nullify if same value & check min,max,valid,invalid
                         2908  7482 ;
            51  87   B0   2908  7483 30$:    MOVW    (R7)+,R1                        ; Get offset + width
                0A   EF   290B  7484          EXTZV   #PRM_OFF_V_WIDTH,-              ; Get width only
        52  51  06        290D  7485                  #PRM_OFF_S_WIDTH,R1,R2          ;   ...
                00   EF   2910  7486          EXTZV   #PRM_OFF_V_VALUE,-              ; Get offset only
        51  51  0A        2912  7487                  #PRM_OFF_S_VALUE,R1,R1          ;   ...
          05 59  03   E0   2915  7488          BBS     #PRM_FLG_V_CDB,R9,40$           ; Br if CDB datum
            51  55   C0   2919  7489          ADDL    R5,R1                           ; Compute offset in UCB
                03   11   291C  7490          BRB     50$                             ; Continue
            51  54   C0   291E  7491 40$:    ADDL    R4,R1                           ; Compute offset in CDB
                         2921  7492 50$:    ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
    2E FB   A7  0C   E0   2921  7493          BBS     #PRM_TYP_V_STRING,-5(R7),100$ ; Br if string data
            58  04   C2   2926  7494          SUBL    #4,R8                           ; Can we get value?
                68   19   2929  7495          BLSS    120$                            ; Br if no - exit
            53  86   D0   292B  7496          MOVL    (R6)+,R3                        ; Get parameter value
                         292E  7497          CASE    R2,TYPE=B,LIMIT=#1,<-           ; Br to handler
                         292E  7498                  60$,-                           ;   Byte value
                         292E  7499                  70$,-                           ;   Word value
                         292E  7500                  80$>                            ;   Longword value
                         2938  7501 ;
                         2938  7502 ; Byte, word, longword value in structure
                         2938  7503 ;
        61  53   90   2938  7504 60$:    MOVB    R3,(R1)                         ; Store byte value
          FF89   31   293B  7505 65$:    BRW     10$                             ; Check remainder
        61  53   B0   293E  7506 70$:    MOVW    R3,(R1)                         ; Store word value
          FF83   31   2941  7507          BRW     10$                             ; Check remainder
        61  55   D0   2944  7508 80$:    MOVL    R5,(R1)                         ; Store longword value
          FF7D   31   2947  7509          BRW     10$                             ; See if more left in P2 buffer
                         294A  7510 ;
                         294A  7511 ; Unknown or invalidated parameter
                         294A  7512 ;
        56  04   C0   294A  7513 90$:    ADDL    #4,R6                           ; Skip value parameter
        58  04   C2   294D  7514          SUBL    #4,R8                           ; Assume a value parameter
            E9   14   2950  7515          BGTR    65$                             ; Br if more
                3F   11   2952  7516          BRB     120$                            ; Else, all done
                         2954  7517 ;
                         2954  7518 ; String parameter in structure
                         2954  7519 ;
        58  02   C2   2954  7520 100$:   SUBL    #2,R8                           ; Can we get string?
                3A   19   2957  7521          BLSS    120$                            ; Br if no - exit
        59  86   3C   2959  7522          MOVZWL  (R6)+,R9                        ; Get string length
        58  59   C2   295C  7523          SUBL    R9,R8                           ; Can we read entire string?
                32   19   295F  7524          BLSS    120$                            ; Br if no - exit
50  0B21  8F   B1   2961  7525          CMPW    #NMA$C_PCLI_DES,R0              ; Is this the destination address?
                05   12   2966  7526          BNEQ    101$                            ; Br if no
              0247   30   2968  7527          BSBW    SET_DESAD                       ; Else, set new destination address
                20   11   296B  7528          BRB     110$                            ; Continue
50  0B04  8F   B1   296D  7529 101$:   CMPW    #NMA$C_PCLI_PHA,R0              ; Is this the physical address?
                05   12   2972  7530          BNEQ    103$                            ; Br if no
```

```
                0230   30  2974  7531          BSBW    SET_PHYAD                       ; Else set new physical address
                  14   11  2977  7532          BRB     110$                            ; Continue
50      0B0F    8F   B1  2979  7533 103$:      CMPW    #NMA$C_PCLI_MCA,R0              ; Is this the multicast address list?
                  05   12  297E  7534          BNEQ    105$                            ; Br if no
                0157   30  2980  7535          BSBW    SET_MULTI                       ; Else, set up new UCB multicast list
                  08   11  2983  7536          BRB     110$                            ; Continue
                  3E   BB  2985  7537 105$:     PUSHR   #^M<R1,R2,R3,R4,R5>             ; Save registers
61      66        59   28  2987  7538          MOVC3   R9,(R6),(R1)                    ; Store string
                  3E   BA  298B  7539          POPR    #^M<R1,R2,R3,R4,R5>             ; Restore registers
                56  59   C0  298D  7540 110$:     ADDL    R9,R6                           ; Point past the string in P2 buffer
                  FF34   31  2990  7541          BRW     10$                             ; Try for more in P2 buffer
                         2993  7542
        07DE    8F   BA  2993  7543 120$:     POPR    #^M<R1,R2,R3,R4,R6,R7,R8,R9,R10> ; Restore registers
                         05  2997  7544          RSB                                     ; Return to caller
```

```
                        2998  7546          .SBTTL  RETURN_P2,  Return UCB/CDB buffer parameters
                        2998  7547
                        2998  7548 ;++
                        2998  7549 ; RETURN_P2 - Return P2 buffer parameters
                        2998  7550 ;
                        2998  7551 ; This routine is called to return the UCB/CDB buffer parameters.
                        2998  7552 ;
                        2998  7553 ; Inputs:
                        2998  7554 ;
                        2998  7555 ;       R3 = IRP address
                        2998  7556 ;       R5 = UCB address
                        2998  7557 ;
                        2998  7558 ; Implicit inputs:
                        2998  7559 ;
                        2998  7560 ;       IRP$L_XQ_P2BUF(R3) = User P2 buffer address
                        2998  7561 ;       IRP$W_XQ_USERSIZ(R3) = User P2 buffer size
                        2998  7562 ;
                        2998  7563 ; Outputs:
                        2998  7564 ;
                        2998  7565 ;       R0 = Size of buffer returned
                        2998  7566 ;       All other registers are preserved.
                        2998  7567 ;
                        2998  7568 ;--
                        2998  7569
                        2998  7570 RETURN_P2::                                 ; Return P2 buffer parameters
    03DE 8F     BB      2998  7571          PUSHR   #^M<R1,R2,R3,R4,R6,R7,R8,R9> ; Save registers
         50     D4      299C  7572          CLRL    R0                          ; Assume no P2 buffer given
56  40 A3     D0      299E  7573          MOVL    IRP$L_XQ_P2BUF(R3),R6       ; Get user P2 buffer address
         03     12      29A2  7574          BNEQ    5$                          ; Br if given
       00BA     31      29A4  7575          BRW     70$                         ; Else, return
                        29A7  7576
54  24 A5     D0      29A7  7577 5$:      MOVL    UCB$L_CRB(R5),R4            ; Get CRB address
54  10 A4     D0      29AB  7578          MOVL    CRB$L_AUXSTRUC(R4),R4       ; Get CDB address
58  38 A3     3C      29AF  7579          MOVZWL  IRP$W_XQ_USERSIZ(R3),R8     ; Get size of user buffer
         56     DD      29B3  7580          PUSHL   R6                          ; Save start of data address
         53     DD      29B5  7581          PUSHL   R3                          ; Save IRP address
51  D6C6 CF     9E      29B7  7582          MOVAB   LINE_PARAM,R1               ; Get address of verification talbe
                        29BC  7583 ;
                        29BC  7584 ; Loop to return next parameter
                        29BC  7585 ;
                        29BC  7586          ASSUME  PRM_W_TYPE EQ 0
    57  81     B0      29BC  7587 10$:     MOVW    (R1)+,R7                    ; Get parameter type code
         03     12      29BF  7588          BNEQ    11$                         ; Br if end of verify table
       0096     31      29C1  7589          BRW     65$                         ; ...
                        29C4  7590
59  57  F000 8F     AB      29C4  7591 11$:     BICW3   #^C<PRM_TYP_M_CODE>,R7,R9 ; Get only the type code
    53  81     9A      29CA  7592          MOVZBL  (R1)+,R3                    ; Get flags byte
    50  81     B0      29CD  7593          MOVW    (R1)+,R0                    ; Get offset + width
                        29D0  7594 ;
                        29D0  7595 ; We will only return NMA$C_PCLI_DES to the SHARED-LIMITED users.
                        29D0  7596 ;
59  0B21 8F     B1      29D0  7597          CMPW    #NMA$C_PCLI_DES,R9          ; Is this a point-to-point parameter?
         07     12      29D5  7598          BNEQ    13$                         ; Br if not
00D4 C5     02     91      29D7  7599          CMPB    #NMA$C_ACC_LIM,UCB$B_XQ_ACC(R5) ; Is this a SHARED-LIMITED user?
         43     12      29DC  7600          BNEQ    50$                         ; Br if not, else return parameter
         0A     EF      29DE  7601 13$:     EXTZV   #PRM_OFF_V_WIDTH,-          ; Get width only
52  50     06              29E0  7602                  #PRM_OFF_S_WIDTH,R0,R2  ;       ...
```

```
                00    EF   29E3  7603            EXTZV   #PRM_OFF_V_VALUE,-          ; Get offset only
          50    50    0A        29E5  7604                  #PRM_OFF_S_VALUE,R0,R0    ;                ...
       05 53    03    E0   29E8  7605            BBS     #PRM_FLG_V_CDB,R3,15$       ; Br if CDB datum
          50    55    C0   29EC  7606            ADDL    R5,R0                       ; Compute offset in UCB
                07    11   29EF  7607            BRB     17$                         ; Continue
                54    D5   29F1  7608 15$:       TSTL    R4                          ; Is CDB given?
                2C    13   29F3  7609            BEQL    50$                         ; Br if no
          50    54    C0   29F5  7610            ADDL    R4,R0                       ; Compute offset in CDB
          58    02    C2   29F8  7611 17$:       SUBL    #2,R8                       ; Any room left in buffer?
                54    19   29FB  7612            BLSS    60$                         ; Br if no - all done
                86    57   B0   29FD  7613            MOVW    R7,(R6)+                    ; Return parameter
    30 FB       A1    0C   E0   2A00  7614            BBS     #PRM_TYP_V_STRING,-5(R1),55$ ; Br if string parameter
          58    04    C2   2A05  7615            SUBL    #4,R8                       ; Any room left?
                47    19   2A08  7616            BLSS    60$                         ; Br if no - all done
                           2A0A  7617            CASE    R2,TYPE=B,LIMIT=#1,<-       ; Br to handler
                           2A0A  7618                    20$,-                       ;   Byte value
                           2A0A  7619                    30$,-                       ;   Word value
                           2A0A  7620                    40$>                        ;   Longword value
                           2A14  7621 ;
                           2A14  7622 ; Byte, word, longword value in structure
                           2A14  7623 ;
                86    60    9A   2A14  7624 20$:       MOVZBL  (R0),(R6)+          ; Store byte value
                08    11   2A17  7625            BRB     50$                         ;
                86    60    3C   2A19  7626 30$:       MOVZWL  (R0),(R6)+          ; Store word value
                03    11   2A1C  7627            BRB     50$                         ;
                86    60    D0   2A1E  7628 40$:       MOVL    (R0),(R6)+          ; Store longword value
                           2A21  7629 50$:       SKIP    PRM_FLG_V_MIN,R3,R1         ; Skip minimum value
                           2A27  7630            SKIP    PRM_FLG_V_MAX,R3,R1         ; Skip maximum value
                           2A2D  7631            SKIP    PRM_FLG_V_INVALID,R3,R1     ; Skip invalid flags
                87    11   2A33  7632            BRB     10$                         ; Try for more parameters
                           2A35  7633 ;
                           2A35  7634 ; String value in structure
                           2A35  7635 ;
          59  0B0F  8F    B1   2A35  7636 55$:       CMPW    #NMA$C_PCLI_MCA,R9          ; Is this the multicast address list?
                05    12   2A3A  7637            BNEQ    57$                         ; Br if no
                   01C7    30   2A3C  7638            BSBW    RETURN_MULTI                ; Else, return multicast address list
                E0    11   2A3F  7639            BRB     50$                         ; Try for more parameters
          58    08    C2   2A41  7640 57$:       SUBL    #8,R8                       ; Any room left?
                0B    19   2A44  7641            BLSS    60$                         ; Br if no - all done
                86    06    9B   2A46  7642            MOVZBW  #6,(R6)+            ; Store string size
                86    80    D0   2A49  7643            MOVL    (R0)+,(R6)+         ; Move data
                86    80    B0   2A4C  7644            MOVW    (R0)+,(R6)+         ;                ...
                D0    11   2A4F  7645            BRB     50$                         ; Try for more parameters
                           2A51  7646
                53    6E    D0   2A51  7647 60$:       MOVL    (SP),R3             ; Get IRP address
    3A A3       0601  8F    B0   2A54  7648            MOVW    #SS$_BUFFEROVF,IRP$W_XQ_STATUS(R3) ; Return error status
                53 8ED0   2A5A  7649 65$:       POPL    R3                          ; Pop stack
          50    56    8E    C3   2A5D  7650            SUBL3   (SP)+,R6,R0         ; Return size of parameters
                03DE  8F    BA   2A61  7651 70$:       POPR    #^M<R1,R2,R3,R4,R6,R7,R8,R9> ; Restore registers
                           05   2A65  7652            RSB                         ; Return to caller
```

```
                        2A66  7654          .SBTTL  VALID_MULTI - VALIDATE THE MULTICAST ADDRESS LIST
                        2A66  7655
                        2A66  7656 ;++
                        2A66  7657 ; VALID_MULTI - VALIDATE THE MULTICAST ADDRESS LIST
                        2A66  7658 ;
                        2A66  7659 ; Functional description:
                        2A66  7660 ;
                        2A66  7661 ; This routine checks all address in the multicast address list to make sure
                        2A66  7662 ; that the logical address bit (lsb) is on.
                        2A66  7663 ;
                        2A66  7664 ; Inputs:
                        2A66  7665 ;
                        2A66  7666 ;       R3 = Size of multicast string list
                        2A66  7667 ;       R4 = CDB address
                        2A66  7668 ;       R5 = UCB address
                        2A66  7669 ;       R6 = Address past multicast strings
                        2A66  7670 ;
                        2A66  7671 ; Outputs:
                        2A66  7672 ;
                        2A66  7673 ;       R0 = Low bit clear if invalid address in list
                        2A66  7674 ;       All other registers are preserved.
                        2A66  7675 ;
                        2A66  7676 ;--
                        2A66  7677
                        2A66  7678 VALID_MULTI:                                 ; Validate the multicast address list
004C 8F     BB          2A66  7679          PUSHR   #^M<R2,R3,R6>               ; Save some registers
50   01     CE          2A6A  7680          MNEGL   #1,R0                       ; Assume success
56   53     C2          2A6D  7681          SUBL    R3,R6                       ; Point back at start of list
53   02     C2          2A70  7682          SUBL    #2,R3                       ; Can we read modifier word?
     26     19          2A73  7683          BLSS    20$                         ; Br if no - error
                        2A75  7684 ;
                        2A75  7685 ; Make sure modifier word is valid - non-zero and less than or equal to
                        2A75  7686 ; NMA$C_LINMC_CAL
                        2A75  7687 ;
                        2A75  7688          ASSUME  NMA$C_LINMC_SET EQ 1
                        2A75  7689          ASSUME  NMA$C_LINMC_CLR EQ 2
                        2A75  7690          ASSUME  NMA$C_LINMC_CAL EQ 3
52   86     B0          2A75  7691          MOVW    (R6)+,R2                    ; Get modifier value
     21     13          2A78  7692          BEQL    20$                         ; Br if zero - illegal
03   52     B1          2A7A  7693          CMPW    R2,#NMA$C_LINMC_CAL         ; Is the modifier okay?
     1C     1A          2A7D  7694          BGTRU   20$                         ; Br if no - error
     1C     13          2A7F  7695          BEQL    30$                         ; Br if "CLEAR ALL" - ignore strings
53   06     C6          2A81  7696          DIVL    #6,R3                       ; Calculate number of strings
     17     13          2A84  7697          BEQL    30$                         ; Br if none
  12 66     E9          2A86  7698 10$:     BLBC    (R6),20$                    ; Br if not a logical address
50   86     D1          2A89  7699          CMPL    (R6)+,R0                    ; Do low order 32 bits = -1?
     05     12          2A8C  7700          BNEQ    15$                         ; Br if no - okay
50   66     B1          2A8E  7701          CMPW    (R6),R0                     ; Do high order 16 bits = -1?
     08     13          2A91  7702          BEQL    20$                         ; Br if yes - illegal
56   02     C0          2A93  7703 15$:     ADDL    #2,R6                       ; Point to next multicast address
  ED 53     F5          2A96  7704          SOBGTR  R3,10$                      ; Loop if more
     02     11          2A99  7705          BRB     30$                         ; Exit with success
                        2A9B  7706
     50     D4          2A9B  7707 20$:     CLRL    R0                          ; Return error
004C 8F     BA          2A9D  7708 30$:     POPR    #^M<R2,R3,R6>               ; Restore registers
            05          2AA1  7709          RSB
```

```
                     2AA2  7711          .SBTTL  VALID_PHYAD - VALIDATE THE PHYSICAL ADDRESS
                     2AA2  7712
                     2AA2  7713 ;++
                     2AA2  7714 ; VALID_PHYAD - VALIDATE THE PHYSICAL ADDRESS
                     2AA2  7715 ;
                     2AA2  7716 ; Functional description:
                     2AA2  7717 ;
                     2AA2  7718 ; This routine checks the physical address to make sure the LSB is clear and
                     2AA2  7719 ; that the modifier word is valid.
                     2AA2  7720 ;
                     2AA2  7721 ; Inputs:
                     2AA2  7722 ;
                     2AA2  7723 ;       R1 = Parameter type code
                     2AA2  7724 ;       R3 = Size of string
                     2AA2  7725 ;       R4 = CDB address
                     2AA2  7726 ;       R5 = UCB address
                     2AA2  7727 ;       R6 = Address past physical address string
                     2AA2  7728 ;
                     2AA2  7729 ; Outputs:
                     2AA2  7730 ;
                     2AA2  7731 ;       R0 = Low bit clear if invalid address in list
                     2AA2  7732 ;       All other registers are preserved.
                     2AA2  7733 ;
                     2AA2  7734 ;--
                     2AA2  7735
                     2AA2  7736 VALID_PHYAD:                            ; Validate the physical address
004C 8F     BB       2AA2  7737          PUSHR   #^M<R2,R3,R6>          ; Save some registers
50   01     CE       2AA6  7738          MNEGL   #1,R0                  ; Assume success
56   53     C2       2AA9  7739          SUBL    R3,R6                  ; Point back at start of list
53   02     C2       2AAC  7740          SUBL    #2,R3                  ; Can we read modifier word?
     22     19       2AAF  7741          BLSS    30$                    ; Br if no - error
                     2AB1  7742
                     2AB1  7743 ;
                     2AB1  7744 ; Make sure modifier word is valid.
                     2AB1  7745 ;
                     2AB1  7746
                     2AB1  7747          ASSUME  NMA$C_LINMC_SET EQ 1
                     2AB1  7748          ASSUME  NMA$C_LINMC_CLR EQ 2
                     2AB1  7749          ASSUME  NMA$C_LINMC_CAL EQ 3
                     2AB1  7750          ASSUME  NMA$C_LINMC_SDF EQ 4
52   86     3C       2AB1  7751          MOVZWL  (R6)+,R2               ; Get modifier value
                     2AB4  7752          $DISPATCH       R2,TYPE=B,-    ; Dispatch on modifier value
                     2AB4  7753                  <-
                     2AB4  7754                  <NMA$C_LINMC_SET 20$>,- ; Set the address
                     2AB4  7755                  <NMA$C_LINMC_CLR 40$>,- ; Clear the address
                     2AB4  7756                  <NMA$C_LINMC_CAL 30$>,- ; 3 - invalid value
                     2AB4  7757                  <NMA$C_LINMC_SDF 10$>,- ; 4 - check it out more
                     2AB4  7758                  >
     11     11       2AC0  7759          BRB     30$                    ; Any other values are invalid
                     2AC2  7760
51   0B04   8F   B1  2AC2  7761 10$:     CMPW    #NMA$C_PCLI_PHA,R1     ; Set to def physical addr requested?
            0A   12  2AC7  7762          BNEQ    30$                    ; Return failure if not
            0A   11  2AC9  7763          BRB     40$                    ; Else, success
                     2ACB  7764
53          06   D1  2ACB  7765 20$:     CMPL    #6,R3                  ; Is string size okay?
            03   12  2ACE  7766          BNEQ    30$                    ; Br if not
      02    66   E9  2AD0  7767          BLBC    (R6),40$               ; Br if a physical address
```

```
                        2AD3  7768
          50    D4      2AD3  7769 30$:    CLRL    R0                      ; Return error
004C      8F    BA      2AD5  7770 40$:    POPR    #^M<R2,R3,R6>           ; Restore registers
                05      2AD9  7771         RSB
```

```
                                                I  3

                     2ADA  7773          .SBTTL  SET_MULTI - SET THE UCB MULTICAST ADDRESS LIST
                     2ADA  7774
                     2ADA  7775 ;++
                     2ADA  7776 ; SET_MULTI - SET THE UCB MULTICAST ADDRESS LIST
                     2ADA  7777 ;
                     2ADA  7778 ; Functional description:
                     2ADA  7779 ;
                     2ADA  7780 ; This routine sets up the multicast addresses in the UCB.
                     2ADA  7781 ;
                     2ADA  7782 ; Inputs:
                     2ADA  7783 ;
                     2ADA  7784 ;       R4 = CDB address
                     2ADA  7785 ;       R5 = UCB address
                     2ADA  7786 ;       R6 = Address of multicast addresses to be set or cleared
                     2ADA  7787 ;       R9 = Size of multicast list
                     2ADA  7788 ;
                     2ADA  7789 ;       IPL = FIPL
                     2ADA  7790 ;
                     2ADA  7791 ; Outputs:
                     2ADA  7792 ;
                     2ADA  7793 ;       R0 = Status return for request
                     2ADA  7794 ;       All registers are preserved.
                     2ADA  7795 ;
                     2ADA  7796 ;--
                     2ADA  7797
                     2ADA  7798 SET_MULTI:                              ; Set up the UCB multicast address list
 024E 8F    BB       2ADA  7799          PUSHR   #^M<R1,R2,R3,R6,R9>    ; Save registers
 59   02    C2       2ADE  7800          SUBL    #2,R9                  ; Can we read the modifier word?
      65    19       2AE1  7801          BLSS    90$                    ; Br if no - exit
 59   06    C6       2AE3  7802          DIVL    #6,R9                  ; Calculate number of addresses
 51   86    3C       2AE6  7803          MOVZWL  (R6)+,R1               ; Get the modifier
                     2AE9  7804          ASSUME  NMA$C_LINMC_SET EQ 1
                     2AE9  7805          ASSUME  NMA$C_LINMC_CLR EQ 2
                     2AE9  7806          ASSUME  NMA$C_LINMC_CAL EQ 3
                     2AE9  7807          CASE    R1,TYPE=B,LIMIT=#1,<-  ; Dispatch on modifier
                     2AE9  7808                  10$,-                  ; Set the address(es)
                     2AE9  7809                  40$,-                  ; Clear the address(es)
                     2AE9  7810                  70$>                   ; Clear ALL addresses
                     2AF3  7811 ;
                     2AF3  7812 ; Set address from list
                     2AF3  7813 ;
      59    D5       2AF3  7814 10$:    TSTL    R9                     ; Any addresses present?
      51    13       2AF5  7815          BEQL    90$                    ; Br if no - exit
 51   86    D0       2AF7  7816 20$:    MOVL    (R6)+,R1               ; Get multicast address
 52   86    B0       2AFA  7817          MOVW    (R6)+,R2               ;       ...
    0169    30       2AFD  7818          BSBW    MATCH_ADDRESS          ; Try to find address in table
   0F 50    E8       2B00  7819          BLBS    R0,30$                 ; Br if present - skip it
      4B    10       2B03  7820          BSBB    FIND_MLTENTRY          ; Find entry in UCB multicast table
   43 50    E9       2B05  7821          BLBC    R0,100$                ; Br if none - leave in error
 83   51    D0       2B08  7822          MOVL    R1,(R3)+               ; Insert new address
 83   52    B0       2B0B  7823          MOVW    R2,(R3)+               ;       ...
 00E5 C5    96       2B0E  7824          INCB    UCB$B_XQ_MULTI(R5)     ; Count one more address
   E2 59    F5       2B12  7825 30$:    SOBGTR  R9,20$                 ; Br if more
      31    11       2B15  7826          BRB     90$                    ; All done
                     2B17  7827 ;
                     2B17  7828 ; Clear address from list
                     2B17  7829 ;
```

```
                                                J 3
                     59   D5   2B17  7830 40$:    TSTL    R9                          ; Any addresses present?
                     2D   13   2B19  7831         BEQL    90$                         ; Br if no - exit
                 51  86   D0   2B1B  7832 50$:    MOVL    (R6)+,R1                    ; Get multicast address
                 52  86   B0   2B1E  7833         MOVW    (R6)+,R2                    ;     ...
                   0145   30   2B21  7834         BSBW    MATCH_ADDRESS               ; Try to find address in table
                  0B 50   E9   2B24  7835         BLBC    R0,60$                      ; Br if not present - skip it
                     83   D4   2B27  7836         CLRL    (R3)+                       ; Mark slot as not in use
                     83   B4   2B29  7837         CLRW    (R3)+                       ;     ...
                00E5 C5   97   2B2B  7838         DECB    UCB$B_XQ_MULTI(R5)          ; Count one less address
                   0042   30   2B2F  7839         BSBW    SQUEEZ_MULTI                ; Squeeze up the multicast list
                  E6 59   F5   2B32  7840 60$:    SOBGTR  R9,50$                      ; Br if more
                     11   11   2B35  7841         BRB     90$                         ; All done
                          2B37  7842 ;
                          2B37  7843 ; Clear all multicast addresses
                          2B37  7844 ;
                00E5 C5   94   2B37  7845 70$:    CLRB    UCB$B_XQ_MULTI(R5)          ; Reset number of multicast addresses
                51   24   9A   2B3B  7846         MOVZBL  #MAX_C_MLT*3,R1             ; Get number of words in multicast list
             52 00E7 C5   9E   2B3E  7847         MOVAB   UCB$G_XQ_MULTI(R5),R2       ; Get address of multicast addresses
                     82   B4   2B43  7848 80$:    CLRW    (R2)+                       ; Init multicast address list
                  FB 51   F5   2B45  7849         SOBGTR  R1,80$                      ; Loop if more
                          2B48  7850
                50   01   9A   2B48  7851 90$:    MOVZBL  S^#SS$_NORMAL,R0            ; Return success
                024E 8F   BA   2B4B  7852 100$:   POPR    #^M<R1,R2,R3,R6,R9>         ; Restore registers
                     05   2B4F  7853         RSB                                 ; Return to caller
                          2B50  7854
                          2B50  7855 ;+
                          2B50  7856 ; FIND_MLTENTRY - FIND EMPTY SLOT IN UCB MULTICAST ADDRESS LIST
                          2B50  7857 ;
                          2B50  7858 ; Inputs:
                          2B50  7859 ;
                          2B50  7860 ;       R5 = UCB address
                          2B50  7861 ;
                          2B50  7862 ; Outputs:
                          2B50  7863 ;
                          2B50  7864 ;       R0 = Status return for request
                          2B50  7865 ;       R3 = Address of available slot if successful
                          2B50  7866 ;-
                          2B50  7867 FIND_MLTENTRY:
                     51   DD   2B50  7868         PUSHL   R1                          ; Save R1
                50   01   CE   2B52  7869         MNEGL   #1,R0                       ; Assume success
             53 00E7 C5   9E   2B55  7870         MOVAB   UCB$G_XQ_MULTI(R5),R3       ; Get address of multicast list
                51   0C   9A   2B5A  7871         MOVZBL  #MAX_C_MLT,R1               ; Get maximum number of addresses
                     83   D5   2B5D  7872 10$:    TSTL    (R3)+                       ; Empty slot?
                     04   12   2B5F  7873         BNEQ    20$                         ; Br if no - skip to next entry
                     63   B5   2B61  7874         TSTW    (R3)                        ; Really?
                     08   13   2B63  7875         BEQL    30$                         ; Br if yes - success
                53   02   C0   2B65  7876 20$:    ADDL    #2,R3                       ; Skip to next address
                   F2 51   F5   2B68  7877         SOBGTR  R1,10$                      ; Br if more to try
                          2B6B  7878
                     50   D4   2B6B  7879         CLRL    R0                          ; Return failure
                53   04   C2   2B6D  7880 30$:    SUBL    #4,R3                       ; Back up pointer
                     51 8ED0   2B70  7881         POPL    R1                          ; Restore registers
                          05   2B73  7882         RSB
                          2B74  7883
                          2B74  7884 ;+
                          2B74  7885 ; SQUEEZ_MULTI - SQUEEZE UP THE MULTICAST ADDRESS LIST
                          2B74  7886 ;
```

```
                     2B74  7887 ; Functional description:
                     2B74  7888 ;
                     2B74  7889 ; The first empty slot is searched for in the multicast address list and
                     2B74  7890 ; the list is squeezed from that point down.
                     2B74  7891 ;
                     2B74  7892 ; Inputs:
                     2B74  7893 ;
                     2B74  7894 ;       R5 = UCB address
                     2B74  7895 ;
                     2B74  7896 ; Ouputs:
                     2B74  7897 ;
                     2B74  7898 ;       None
                     2B74  7899 ;
                     2B74  7900 ;--
                     2B74  7901 SQUEEZ_MULTI:                                ; Squeeze down the multicast address list
                     2B74  7902         PUSHQ   R3                           ; Save R3, R4
54   00E5 C5   9A    2B77  7903         MOVZBL  UCB$B_XQ_MULTI(R5),R4        ; Get number of entries in list
           25   13    2B7C  7904         BEQL    40$                          ; Br if none
53   00E7 C5   9E    2B7E  7905         MOVAB   UCB$G_XQ_MULTI(R5),R3        ; Get address of multicast list
                     2B83  7906 ;
                     2B83  7907 ; Search for empty slot in list
                     2B83  7908 ;
           63   D5   2B83  7909 10$:    TSTL    (R3)                         ; Is this the empty slot
           05   12   2B85  7910         BNEQ    20$                          ; Br if no
        04 A3   B5   2B87  7911         TSTW    4(R3)                        ; Really?
           08   13   2B8A  7912         BEQL    30$                          ; Br if yes
     53    06   C0   2B8C  7913 20$:    ADDL    #6,R3                        ; Skip to next entry in list
       F1  54   F5   2B8F  7914         SOBGTR  R4,10$                       ; Loop if more possiblities
           0F   11   2B92  7915         BRB     40$                          ; Else, exit
                     2B94  7916 ;
                     2B94  7917 ; Empty slot found - put the squeeze on.
                     2B94  7918 ;
83   06 A3   D0   2B94  7919 30$:    MOVL    6(R3),(R3)+                  ; Squeeze the list
83   06 A3   B0   2B98  7920         MOVW    6(R3),(R3)+                  ;       ...
     F5 54   F5   2B9C  7921         SOBGTR  R4,30$                       ; Loop if more to go
        83   D4   2B9F  7922         CLRL    (R3)+                        ; Zero the last entry
        83   B4   2BA1  7923         CLRW    (R3)+                        ;       ...
                     2BA3  7924 40$:    POPQ    R3                           ; Restore R3, R4
                05   2BA6  7925         RSB                                  ; Return to caller
```

```
                                2BA7  7927          .SBTTL  SET_PHYAD - SET THE PHYSICAL ADDRESS
                                2BA7  7928          .SBTTL  SET_DESAD - SET THE DESTINATION ADDRESS
                                2BA7  7929 ;++
                                2BA7  7930 ; SET_PHYAD - SET THE PHYSICAL ADDRESS
                                2BA7  7931 ; SET_DESAD - SET THE DESTINATION ADDRESS
                                2BA7  7932 ;
                                2BA7  7933 ; Functional description:
                                2BA7  7934 ;
                                2BA7  7935 ; This routine sets up the physical address in the CDB.
                                2BA7  7936 ;
                                2BA7  7937 ; Inputs:
                                2BA7  7938 ;
                                2BA7  7939 ;       R4 = CDB address
                                2BA7  7940 ;       R5 = UCB address
                                2BA7  7941 ;       R6 = Address of physical address to be set or cleared
                                2BA7  7942 ;       R9 = Size of the string
                                2BA7  7943 ;
                                2BA7  7944 ;       IPL = FIPL
                                2BA7  7945 ;
                                2BA7  7946 ; Outputs:
                                2BA7  7947 ;
                                2BA7  7948 ;       R0 = destroyed.
                                2BA7  7949 ;       All other registers are preserved.
                                2BA7  7950 ;
                                2BA7  7951 ;--
                                2BA7  7952          .ENABL  LSB
                                2BA7  7953 SET_PHYAD:                           ; Set up the CDB physical address
        02C2 8F     BB          2BA7  7954          PUSHR   #^M<R1,R6,R7,R9>    ; Save registers
    57  00DE C5     9E          2BAB  7955          MOVAB   UCB$G_XQ_PHA(R5),R7 ; Get address of UCB cell
            09      11          2BB0  7956          BRB     10$                 ; Join common code
                                2BB2  7957 SET_DESAD:
        02C2 8F     BB          2BB2  7958          PUSHR   #^M<R1,R6,R7,R9>    ; Save registers
    57  00CC C5     9E          2BB6  7959          MOVAB   UCB$G_XQ_DES(R5),R7 ; Get address of UCB cell
        59  02      C2          2BBB  7960 10$:     SUBL    #2,R9               ; Can we read the modifier word?
            41      19          2BBE  7961          BLSS    50$                 ; Br if no - exit
        59  06      C6          2BC0  7962          DIVL    #6,R9               ; Calculate number of addresses
        51  86      3C          2BC3  7963          MOVZWL  (R6)+,R1            ; Get the modifier
                                2BC6  7964
                                2BC6  7965          ASSUME  NMA$C_LINMC_SET EQ 1
                                2BC6  7966          ASSUME  NMA$C_LINMC_CLR EQ 2
                                2BC6  7967          ASSUME  NMA$C_LINMC_CAL EQ 3
                                2BC6  7968          ASSUME  NMA$C_LINMC_SDF EQ 4
                                2BC6  7969          $DISPATCH       R1,TYPE=B,- ; Dispatch on modifier value
                                2BC6  7970                  <-
                                2BC6  7971                  <NMA$C_LINMC_SET 20$>,- ; Set the address
                                2BC6  7972                  <NMA$C_LINMC_CLR 30$>,- ; Clear the address
                                2BC6  7973                  <NMA$C_LINMC_CAL 50$>,- ; 3 - invalid value
                                2BC6  7974                  <NMA$C_LINMC_SDF 40$>,- ; Set physical as DECnet default address
                                2BC6  7975                  >
            2D      11          2BD2  7976          BRB     50$                 ; Any other values are invalid
                                2BD4  7977
                                2BD4  7978 ;
                                2BD4  7979 ; Set physical address
                                2BD4  7980 ;
        59  D5      13          2BD4  7981 20$:     TSTL    R9                  ; Any addresses present?
            29      13          2BD6  7982          BEQL    50$                 ; Br if no - exit
    87  86  D0      2BD8  7983          MOVL    (R6)+,(R7)+                     ; Set new address
```

```
                67   86   B0  2BDB  7984          MOVW    (R6)+,(R7)              ;                ...
                     21   11  2BDE  7985          BRB     50$                     ; All done
                               2BE0  7986 ;
                               2BE0  7987 ; Clear address from list
                               2BE0  7988 ;
                87   01   CE  2BE0  7989 30$:     MNEGL   #1,(R7)+                ; Reset address
                67   01   AE  2BE3  7990          MNEGW   #1,(R7)                 ;                ...
                     19   11  2BE6  7991          BRB     50$                     ; All done
                               2BE8  7992
                               2BE8  7993 ;
                               2BE8  7994 ; Set the physical address (Assume entered from SET_PHYAD) to the DECNET
                               2BE8  7995 ; default address. The PHA address has been pre-set with -1, which implies
                               2BE8  7996 ; using the HWA.
                               2BE8  7997 ;
                     00   E0  2BE8  7998 40$:     BBS     #CDB_STS_V_INITED,-      ; Br if XQ device is already inited
                13 024A   C4  2BEA  7999                 CDB_B_STS(R4),50$        ;    use HWA
           51   00000000'GF   B0  2BEE  8000          MOVW    G^SCS$GB_SYSTEMID,R1    ; Else, get SCSSYSTEMID parameter
                     0A   13  2BF5  8001          BEQL    50$                     ; Br if none, use HWA
           87   000400AA   8F   D0  2BF7  8002          MOVL    #^X0004C0AA,(R7)+       ; Set common low longword of addr
                67   51   B0  2BFE  8003          MOVW    R1,(R7)                 ; Set unique high word of address
                               2C01  8004
                02C2   8F   BA  2C01  8005 50$:     POPR    #^M<R1,R6,R7,R9>        ; Restore registers
                     05   2C05  8006          RSB                             ; Return to caller
                               2C06  8007          .DSABL  LSB
```


XQD
Sym
UCB
UCB
UCB
UCB
UCB
UCB
UCB
UCB
UCB
UCB
UCB
UCB
UCB
UCB
UCB
UCB
UNI
UV1
UV1
UV1
VAS
VAS
VAS
VAL
VAL
VAL
VEC
VEC
VEC
VEC
VEC
VEC
VEC
VEC
XBU
XBU
XBU
XBU
XBU
XBU
XMS
XMS
XMS
XMS
XMS
XMS
XMT
XMT
XMT
XMT
XMT
XMT
XMT
XMT
XMT

```
                                2C06  8009          .SBTTL  RETURN_MULTI - RETURN THE MULTICAST ADDRESS LIST
                                2C06  8010
                                2C06  8011 ;++
                                2C06  8012 ; RETURN_MULTI - RETURN THE MULTICAST ADDRESS LIST
                                2C06  8013 ;
                                2C06  8014 ; Functional description:
                                2C06  8015 ;
                                2C06  8016 ; This routine returns all the multicast addresses in the UCB list.
                                2C06  8017 ;
                                2C06  8018 ; Inputs:
                                2C06  8019 ;
                                2C06  8020 ;       R4 = CDB address
                                2C06  8021 ;       R5 = UCB address
                                2C06  8022 ;       R6 = Address where to store multicast address list
                                2C06  8023 ;       R8 = Size left in output buffer
                                2C06  8024 ;
                                2C06  8025 ; Outputs:
                                2C06  8026 ;
                                2C06  8027 ;       R6 = Address past stored multicast address list
                                2C06  8028 ;       All other registers are preserved.
                                2C06  8029 ;
                                2C06  8030 ;--
                                2C06  8031
                                2C06  8032 RETURN_MULTI:                                ; Return the multicast address list
                                2C06  8033          PUSHQ   R1                          ; Save R1, R2
52  00E5 C5   9A                2C09  8034          MOVZBL  UCB$B_XQ_MULTI(R5),R2       ; Set number of multicast addresses
51  00E7 C5   9E                2C0E  8035          MOVAB   UCB$G_XQ_MULTI(R5),R1       ; Point to start of multicast lists
    58   02   C2                2C13  8036          SUBL    #2,R8                       ; Can string size fit?
         1E   19                2C16  8037          BLSS    40$                         ; Br if no - return
    56   02   C0                2C18  8038          ADDL    #2,R6                       ; Skip size field of return data
         56   DD                2C1B  8039          PUSHL   R6                          ; Save start address of string
    58   06   C2                2C1D  8040 10$:     SUBL    #6,R8                       ; Room left in buffer?
         09   19                2C20  8041          BLSS    30$                         ; Br if no .. exit now
    86   81   D0                2C22  8042          MOVL    (R1)+,(R6)+                 ; Return address
    86   81   B0                2C25  8043          MOVW    (R1)+,(R6)+                 ;    ...
      F2 52   F5                2C28  8044          SOBGTR  R2,10$                      ; Br if more possibles
                                2C2B  8045
         51 8ED0                2C2B  8046 30$:     POPL    R1                          ; Restore string address
52  56   51   C3                2C2E  8047          SUBL3   R1,R6,R2                    ; Get string size
  FE A1   52   B0               2C32  8048          MOVW    R2,-2(R1)                   ; Return string size
                                2C36  8049 40$:     POPQ    R1                          ; Restore R1, R2
                   05           2C39  8050          RSB                                 ; Return to caller
```

```
                     2C3A  8052          .SBTTL  MATCH_MULTI - CHECK MULTICAST ADDRESS
                     2C3A  8053
                     2C3A  8054 ;++
                     2C3A  8055 ; MATCH_MULTI - CHECK MULTICAST ADDRESS
                     2C3A  8056 ;
                     2C3A  8057 ; Functional description:
                     2C3A  8058 ;
                     2C3A  8059 ; This routine returns success if the unit is in promiscous mode, or the
                     2C3A  8060 ; recognizes all multicast address or the multicast address in the buffer
                     2C3A  8061 ; matches a multicast address in the unit's multicast address list.
                     2C3A  8062 ;
                     2C3A  8063 ; Inputs:
                     2C3A  8064 ;
                     2C3A  8065 ;       R2 = Receive buffer
                     2C3A  8066 ;       R4 = CDB address
                     2C3A  8067 ;       R5 = UCB address
                     2C3A  8068 ;
                     2C3A  8069 ; Outputs:
                     2C3A  8070 ;
                     2C3A  8071 ;       R0 = Status return for request
                     2C3A  8072 ;       R1 = Destroyed
                     2C3A  8073 ;       All other registers are preserved.
                     2C3A  8074 ;
                     2C3A  8075 ;--
                     2C3A  8076
                     2C3A  8077 MATCH_MULTI:                            ; Find multicast addres in UCB
                     2C3A  8078          PUSHQ   R2                     ; Save R2, R3
                  50 D4  2C3D  8079          CLRL    R0                     ; Assume failure
                     2C3F  8080
                     2C3F  8081          ASSUME  UCB$V_XQ_INITED  EQ  0
                  22 68 A5  E9  2C3F  8082          BLBC    UCB$W_DEVSTS(R5),10$   ; Br if PROTOCOL TYPE is not valid
                        50  D6  2C43  8083          INCL    R0                     ; Assume success
                            2C45  8084          ASSUME  NMA$C_STATE_ON EQ 0
                            2C45  8085          ASSUME  NMA$C_STATE_OFF EQ 1
                1B 00DA C5  E9  2C45  8086          BLBC    UCB$B_XQ_PRM(R5),10$   ; Br if promiscuous mode - OKAY
                16 00DB C5  E9  2C4A  8087          BLBC    UCB$B_XQ_MLT(R5),10$   ; Br if all multicasts are enabled
                51    38 A2 7D  2C4F  8088          MOVQ    CXB$G_R_DEST(R2),R1    ; Get multicast address
             51 FFFFFFFF 8F  D1  2C53  8089          CMPL    #BRDCSTT,R1            ; Is this the broadcast address?
                        07  12  2C5A  8090          BNEQ    5$                     ; Br if no
                52    FFFF 8F  B1  2C5C  8091          CMPW    #BRDCST2,R2            ; Really?
                        02  13  2C61  8092          BEQL    10$                    ; Br if yes - everybody get this one
                        04  10  2C63  8093 5$:       BSBB    MATCH_ADDRESS          ; Else, find exact match
                            2C65  8094 10$:      POPQ    R2                     ; Restore R2, R3
                        05  2C68  8095          RSB                            ; Return to caller
```

```
                     2C69  8097          .SBTTL  MATCH_ADDRESS - FIND A MATCH ON A MULTICAST ADDRESS
                     2C69  8098
                     2C69  8099 ;++
                     2C69  8100 ; MATCH_ADDRESS - FIND A MATCH ON A MULTICAST ADDRESS
                     2C69  8101 ;
                     2C69  8102 ; Functional description:
                     2C69  8103 ;
                     2C69  8104 ; This routine searches the UCB multicast address list for a match on a
                     2C69  8105 ; multicast address.
                     2C69  8106 ;
                     2C69  8107 ; Inputs:
                     2C69  8108 ;
                     2C69  8109 ;       R1 = Low 32 bits of 48 bit multicast address to match
                     2C69  8110 ;       R2 = High 16 bits of 48 bit multicast address to match
                     2C69  8111 ;       R4 = CDB address
                     2C69  8112 ;       R5 = UCB address
                     2C69  8113 ;
                     2C69  8114 ; Outputs:
                     2C69  8115 ;
                     2C69  8116 ;       R0 = Status return for request
                     2C69  8117 ;       R3 = Address of slot in multicast address list
                     2C69  8118 ;       All other registers are preserved.
                     2C69  8119 ;
                     2C69  8120 ;--
                     2C69  8121
                     2C69  8122 MATCH_ADDRESS:                          ; Find multicast address in UCB
          54   DD    2C69  8123          PUSHL   R4                      ; Save R4
     50   01   9A    2C6B  8124          MOVZBL  S^#SS$_NORMAL,R0        ; Assume success
54   00E5 C5   9A    2C6E  8125          MOVZBL  UCB$B_XQ_MULTI(R5),R4   ; Set number of multicast addresses
53   00E7 C5   9E    2C73  8126          MOVAB   UCB$G_XQ_MULTI(R5),R3   ; Point to start of multicast lists
     51   83   D1    2C78  8127 10$:     CMPL    (R3)+,R1                ; Is this a match?
          05   12    2C7B  8128          BNEQ    20$                     ; Br if no - skip to next
     52   63   B1    2C7D  8129          CMPW    (R3),R2                 ; Is it really?
          08   13    2C80  8130          BEQL    30$                     ; Br if yes - all done
     53   02   C0    2C82  8131 20$:     ADDL    #2,R3                   ; Skip to next entry
       F0 54   F5    2C85  8132          SOBGTR  R4,10$                  ; Br if more in list
                     2C88  8133
          50   D4    2C88  8134          CLRL    R0                      ; Return failure
     53   04   C2    2C8A  8135 30$:     SUBL    #4,R3                   ; Backup pointer
          54 8ED0    2C8D  8136          POPL    R4                      ; Restore R4
                05   2C90  8137          RSB                             ; Return to caller
```

```
                        2C91  8139          .SBTTL  POKE_USER - DELIVER ATTENTION ASTS
                        2C91  8140
                        2C91  8141 ;++
                        2C91  8142 ; POKE_USER - Deliver attention AST
                        2C91  8143 ;
                        2C91  8144 ; Functional description:
                        2C91  8145 ;
                        2C91  8146 ; This routine is used to deliver an attention AST if one has been
                        2C91  8147 ; requested.
                        2C91  8148 ;
                        2C91  8149 ; Inputs:
                        2C91  8150 ;
                        2C91  8151 ;       R5 = UCB address
                        2C91  8152 ;
                        2C91  8153 ; Outputs:
                        2C91  8154 ;
                        2C91  8155 ;       R0 = Low bit clear only if user is not notified
                        2C91  8156 ;       R1-R3 are destroyed.
                        2C91  8157 ;
                        2C91  8158 ;--
                        2C91  8159
                        2C91  8160 POKE_USER:                                   ; Poke user process
                        2C91  8161          DSBINT  UCB$B_FIPL(R5)              ; Sync access to UCB
              7E    D4  2C98  8162          CLRL    -(SP)                       ; Assume failure
    51  00C0  C5    9E  2C9A  8163          MOVAB   UCB$L_XQ_AST(R5),R1         ; Get AST listhead
              61    D5  2C9F  8164          TSTL    (R1)                        ; Empty?
              1C    13  2CA1  8165          BEQL    30$                         ; Branch if yes
              6E    D6  2CA3  8166          INCL    (SP)                        ; Indicate success
              54    DD  2CA5  8167          PUSHL   R4                          ; Save R4
        54    51    D0  2CA7  8168          MOVL    R1,R4                       ; Copy listhead address
        51    61    D0  2CAA  8169 10$:     MOVL    (R1),R1                     ; Address a block
              07    13  2CAD  8170          BEQL    20$                         ; Branch if done
           44 A5    D0  2CAF  8171          MOVL    UCB$L_DEVDEPEND(R5),-       ; Change parameter
           1C A1        2CB2  8172                  ACB$L_KAST+4(R1)            ;    return status
              F4    11  2CB4  8173          BRB     10$                         ; Continue thru AST blocks
00000000'GF     16  2CB6  8174 20$:     JSB     G^COM$DELATTNAST            ; Deliver the AST's
              54 8ED0   2CBC  8175          POPL    R4                          ; Restore R4
                        2CBF  8176
              50 8ED0   2CBF  8177 30$:     POPL    R0                          ; Return success indicator
                        2CC2  8178          ENBINT                              ; Restore IPL
                    05  2CC5  8179          RSB                                 ; Return to caller
```

```
                        2CC6  8181          .SBTTL  MATCH_PROTYP - Match protocol type
                        2CC6  8182          .SBTTL  MATCH_PROMTYP - Find the promiscuous user
                        2CC6  8183
                        2CC6  8184 ;++
                        2CC6  8185 ; MATCH_PROTYP - Match protocol type
                        2CC6  8186 ; MATCH_PROMTYP - Find the promiscuous user
                        2CC6  8187 ;
                        2CC6  8188 ; This routine checks for a match of a protocol type against that in
                        2CC6  8189 ; existing UCB's.
                        2CC6  8190 ;
                        2CC6  8191 ; Inputs:
                        2CC6  8192 ;
                        2CC6  8193 ;       R1 = word of protocol type
                        2CC6  8194 ;       R4 = CDB address
                        2CC6  8195 ;
                        2CC6  8196 ; Outputs:
                        2CC6  8197 ;
                        2CC6  8198 ;       R0 = LBS=> match; LBC=> no match
                        2CC6  8199 ;       R5 = UCB address on success
                        2CC6  8200 ;
                        2CC6  8201 ;-
                        2CC6  8202
                        2CC6  8203 MATCH_PROTYP:                                ; Match protocol type
              50   D4   2CC6  8204          CLRL    R0                         ; Assume failure
        55  0118 C4   D0   2CC8  8205          MOVL    CDB_L_UCB0(R4),R5          ; Get first UCB address
                18   13   2CCD  8206          BEQL    20$                        ; Br if not inited - yet
          55    30 A5   D0   2CCF  8207 10$:   MOVL    UCB$L_LINK(R5),R5          ; Get next UCB address
                12   13   2CD3  8208          BEQL    20$                        ; If EQL no match
                             2CD5  8209          ASSUME  UCB$V_XQ_INITED  EQ  0
             F6 68 A5   E9   2CD5  8210          BLBC    UCB$W_DEVSTS(R5),10$       ; Br if PROTOCOL TYPE is not valid
                             2CD9  8211
                             2CD9  8212          ASSUME  NMA$C_STATE_ON EQ 0
                             2CD9  8213          ASSUME  NMA$C_STATE_OFF EQ 1
                             2CD9  8214
           F1 00DA C5   E9   2CD9  8215          BLBC    UCB$B_XQ_PRM(R5),10$       ; Skip if PROMISCUOUS user
        00CA C5     51   B1   2CDE  8216          CMPW    R1,UCB$W_XQ_PROTYP(R5)     ; Match?
                    EA   12   2CE3  8217          BNEQ    10$                        ; If NEQ no - loop
                    50   D6   2CE5  8218 15$:     INCL    R0                         ; Return success
                    05   2CE7  8219 20$:     RSB                                ; Done
                         2CE8  8220
                         2CE8  8221 MATCH_PROMTYP:
              50    01   9A   2CE8  8222          MOVZBL  #1,R0                      ; Assume success
        55  0214 C4   D0   2CEB  8223          MOVL    CDB_L_PRMUSER(R4),R5       ; Get PROMISCUOUS user's UCB address
                    02   12   2CF0  8224          BNEQ    10$                        ; Br if present
                    50   D4   2CF2  8225          CLRL    R0                         ; Else, return error
                    05   2CF4  8226 10$:     RSB                                ; Return to caller
                         2CF5  8227
                         2CF5  8228
                         2CF5  8229 XQ_END::
                         2CF5  8230          .END
```

```
$$$                     = 00000020 R     02
$$$FLG                  = 00000013
$$$MAP                  = 00000000
$$$OFF                  = 000000E4
$$$SIZ                  = 00000008
$$$TYP                  = 00000429
$$$WID                  = 00000002
$$BASE                  = 00000001
$$DISPL                 = 00000008
$$GENSW                 = 00000001
$$HIGH                  = 00000007
$$LIMIT                 = 00000006
$$LOW                   = 00000001
$$MNSW                  = 00000001
$$MXSW                  = 00000001
$$OP                    = 00000002
ABORTIO                   00000346 R     03
ABORTIO_BR                0000028E R     03
ABORT_IRP                 00000CD3 R     03
ABORT_PKT                 00001BC2 R     03
ACB$L_KAST              = 00000018
ACCESS                    00000DF0 R     03
ADDRCVLIST                00001440 RG    03
ADD_MULTI                 000026B1 R     03
ALLOC_CDB                 00001F81 R     03
ALLOC_P2BUF               00000E09 R     03
ALT_START                 0000074C RG    03
ASSEM_PKTS                00001C33 RG    03
AT$_UBA                 = 00000001
BAD_PARAM_TBL             000001A6 R     03
BLD_IRP                   00002380 R     03
BLD_STOP_IRP              00002309 R     03
BLD_STRT_IRP              00002342 R     03
BLK_B_SPARE               0000000B
BLK_B_TYPE                0000000A
BLK_C_HEADER              0000000C
BLK_L_LINK                00000000
BLK_T_DATA                0000000C
BLK_W_SIZE                00000008
BRDCST1                 = FFFFFFFF
BRDCST2                 = 0000FFFF
BUG$_NOBUFPCKT            ******** X     03
BUG$_UNSUPRTCPU           ******** X     03
CAN$C_CANCEL            = 00000000
CAN$C_DASSGN            = 00000001
CANCEL                    00002472 RG    03
CCB$L_UCB               = 00000000
CDB_B_AQUOTA              000002F2  G
CDB_B_CON                 0000024D  G
CDB_B_DIAG1               0000011D  G
CDB_B_FIPL                0000000B  G
CDB_B_LASTRCV             0000010C  G
CDB_B_LASTXMT             0000010D  G
CDB_B_MLT                 0000024C  G
CDB_B_MLTTBL              00000261  G
CDB_B_MQUOTA              000002F3  G
CDB_B_MULTI               00000260  G
CDB_B_NEXTRCV             00000019  G
CDB_B_NEXTXMT             00000018  G
CDB_B_PRM                 0000024B  G
CDB_B_RCVCNT              0000010E  G
CDB_B_RCVMAP              0000001A  G
CDB_B_SETPRM              0000024D  G
CDB_B_SPARE               0000011C  G
CDB_B_STS                 0000024A  G
CDB_B_TIM_XMT             0000020E  G
CDB_B_TYPE                0000000A  G
CDB_B_UNTCNT              0000020F  G
CDB_B_XMTCNT              0000010F  G
CDB_B_XMTMAP              0000001B  G
CDB_C_ABORTS            = 00000001
CDB_C_LENGTH              000002F4  G
CDB_C_MAPPED            = 000000A8
CDB_C_QUEUES            = 00000006
CDB_C_SETPRM            = 00000001
CDB_C_ZERO                0000020E  G
CDB_G_COUNTER             00000120  G
CDB_G_HWA                 00000254  G
CDB_G_MAPPED              00000166  G
CDB_G_MLTTBL              000002AA  G
CDB_G_MULTI               00000262  G
CDB_G_PHA                 0000024E  G
CDB_G_PHYADR              0000025A  G
CDB_G_RRING               00000166  G
CDB_G_XRING               000001D2  G
CDB_L_BIDCTR              0000014A  G
CDB_L_BRCCTR              0000012E  G
CDB_L_BS1CTR              00000146  G
CDB_L_BSMCTR              00000142  G
CDB_L_BSNCTR              0000014E  G
CDB_L_CSR                 00000010  G
CDB_L_DBRCTR              00000122  G
CDB_L_DBSCTR              0000013A  G
CDB_L_DEVDEPEND           00000114  G
CDB_L_FPC                 0000000C  G
CDB_L_FQBL                00000004  G
CDB_L_FQFL                00000000  G
CDB_L_FR3                 00000010  G
CDB_L_FR4                 00000014  G
CDB_L_MBLCTR              00000126  G
CDB_L_MBSCTR              0000013E  G
CDB_L_MBYCTR              00000132  G
CDB_L_MSNCTR              00000152  G
CDB_L_PRMUSER             00000214  G
CDB_L_RCVMAP              0000001C  G
CDB_L_RCV_PA              000000AC  G
CDB_L_RCV_VA              000000C4  G
CDB_L_RINGMAP             00000162  G
CDB_L_RRINGPA             00000044  G
CDB_L_RRINGVA             0000007C  G
CDB_L_TQE                 00000218  G
CDB_L_UCBO                00000118  G
CDB_L_UV1BUF              00000210  G
CDB_L_XMTMAP              00000038  G
```

```
CDB_L_XMT_PA                    000000C0  G
CDB_L_XMT_VA                    000000D8  G
CDB_L_XRINGPA                   00000068  G
CDB_L_XRINGVA                   0000009C  G
CDB_MOD_M_MULTI               = 00000001  G
CDB_MOD_M_PROM                = 00000002  G
CDB_MOD_V_MULTI               = 00000000  G
CDB_MOD_V_PROM                = 00000001  G
CDB_Q_INPUT                     000000E4  G
CDB_Q_POST                      00000104  G
CDB_Q_QUEUES                    000000DC  G
CDB_Q_RCVBUF                    000000F4  G
CDB_Q_RCVPND                    000000FC  G
CDB_Q_XMTPND                    000000EC  G
CDB_Q_XMTREQ                    000000DC  G
CDB_STS_M_ERR                 = 00000010  G
CDB_STS_M_FORK_PEND           = 00000004  G
CDB_STS_M_INITED              = 00000001  G
CDB_STS_M_RUN                 = 00000002  G
CDB_STS_M_SETUP               = 00000020  G
CDB_STS_M_TIMER               = 00000008  G
CDB_STS_V_ERR                 = 00000004  G
CDB_STS_V_FORK_PEND           = 00000002  G
CDB_STS_V_INITED              = 00000000  G
CDB_STS_V_RUN                 = 00000001  G
CDB_STS_V_SETUP               = 00000005  G
CDB_STS_V_TIMER               = 00000003  G
CDB_W_BSZ                       00000110  G
CDB_W_CDCCTR                    0000015A  G
CDB_W_DIAG2                     0000011E  G
CDB_W_LBECTR                    00000138  G
CDB_W_MODE                      00000248  G
CDB_W_OVRCTR                    00000136  G
CDB_W_QUOTA                     00000112  G
CDB_W_RFLCTR                    0000012C  G
CDB_W_RFLMAP                    0000012A  G
CDB_W_SBUCTR                    0000015E  G
CDB_W_SFLCTR                    00000158  G
CDB_W_SFLMAP                    00000156  G
CDB_W_SIZE                      00000008  G
CDB_W_UBUCTR                    00000160  G
CDB_W_UFDCTR                    0000015C  G
CDB_W_ZERO                      00000120  G
CHANGE_PARAM                    000028A8  R     03
CHECKER                         0000250B  R     03
CHECKPKT                        00002525  R     03
CHECK_BUFS                      00000DCD  R     03
CHECK_P1                        00000DF6  R     03
CHECK_P2                        00000DCF  R     03
CHECK_PARAM                     00000BA1  R     03
CHECK_QUOTA                     00000A3F  R     03
CHECK_SHR                       00000730  R     03
CHECK_SRC                       00001A39  R     03
CHMODE                          00000EBF  R     03
CIRCUIT_PARAM                   000000FE  R     03
CIRC_CTR                        00000159  R     03
CIRC_CTR_BUFSIZ               = 00000020
CIRC_CTR_SIZE                 = 00000006
CLEANUP_SHR                     00002396  RG    03
CLONED_UCB                      000001B7  RG    03
COM$DECATTNAST                  ********    X   03
COM$DRVDEALMEM                  ********    X   03
COM$FLUSHATTNS                  ********    X   03
COM$POST                        ********    X   03
COM$SETATTNAST                  ********    X   03
CONTROL_INIT                    000001B6  RG    03
COPY_RCV                        00001A46  R     03
CRB$C_AUXSTRUC                = 00000010
CRB$L_INTD                    = 00000024
CSR                             0000000E  G
CXB$B_CODE                    = 0000000B
CXB$B_R_FLAGS                   0000000B  G
CXB$B_TYPE                    = 0000000A
CXB$B_XQ_FUNC                   00000020  G
CXB$B_XQ_RING                   00000023  G
CXB$B_XQ_SLOT                   00000022  G
CXB$C_HEADER                  = 00000048
CXB$C_TRAILER                 = 00000004
CXB$G_R_DEST                    00000038  G
CXB$G_R_SRC                     0000003E  G
CXB$L_BC                      = 00000004
CXB$L_END_ACTION              = 0000001C
CXB$L_FL                      = 00000000
CXB$L_IRP                     = 00000014
CXB$L_LINK                    = 00000010
CXB$L_SPARE0                  = 00000024
CXB$L_SPARE1                  = 00000020
CXB$L_T_IRP                     00000024  G
CXB$L_T_UCB                     00000024  G
CXB$Q_STATION                 = 00000028
CXB$T_R_DATA                    00000038  G
CXB$T_R_USERDAT                 00000046  G
CXB$T_T_DATA                    0000003A  G
CXB$W_BCNT                    = 0000001A
CXB$W_BOFF                    = 00000018
CXB$W_LENGTH                  = 0000000C
CXB$W_R_NCHAIN                  0000001C  G
CXB$W_R_PTYPE                   00000044  G
CXB$W_R_SIZE                    00000046  G
CXB$W_R_STS                     00000014  G
CXB$W_SIZE                    = 00000008
CXB$W_XQ_RID                    00000022  G
CXB_CHECKER                     00002548  R     03
CXB_CHECKPKT                    00002589  R     03
DC$_SCOM                      = 00000020
DDB$L_DDT                     = 0000000C
DDB$L_UCB                     = 00000004
DELETE_BLOCK                    00002339  R     03
DELETE_SHR                      0000240A  RG    03
DEV$M_AVL                     = 00040000
DEV$M_IDV                     = 04000000
DEV$M_NET                     = 00002000
DEV$M_ODV                     = 08000000
DEV$M_SHR                     = 00010000
```

```
DEV_TIMEOUT                 00001F57 RG    03
DIAG_B_SPARE                0000000B
DIAG_B_TYPE                 0000000A
DIAG_C_EXTRA              = 00000006
DIAG_C_LENGTH               0000003E
DIAG_G_DEST                 00000030
DIAG_G_HWA                  0000002A
DIAG_G_SRC                  00000036
DIAG_L_BUFFER               00000004
DIAG_L_DATA                 00000000
DIAG_L_DEPEND               00000024
DIAG_L_ERRS                 0000001C
DIAG_L_EXTRA                00000020
DIAG_Q_FINISH               00000014
DIAG_Q_START                0000000C
DIAG_T_DATA                 0000000C
DIAG_T_RDATA                00000030
DIAG_W_CSR                  00000024
DIAG_W_ERR                  00000026
DIAG_W_ERR2                 00000028
DIAG_W_SIZE                 00000008
DIAG_W_TYPE                 0000003C
DNI_C_TIM                 = 0000000A
DNI_TIM                   = 00000006
DPT$B_FLAGS               = 0000000D
DPT$C_LENGTH              = 00000038
DPT$C_VERSION             = 00000004
DPT$INITAB                  00000038 R     02
DPT$M_NOUNLOAD            = 00000004
DPT$REINITAB                0000009F R     02
DPT$TAB                     00000000 R     02
DSC$A_POINTER             = 00000004
DT$_DEQNA                 = 00000016
DYN$C_BUFIO               = 00000013
DYN$C_CDB                 = 00000033
DYN$C_CRB                 = 00000005
DYN$C_CXB                 = 0000001B
DYN$C_DDB                 = 00000006
DYN$C_DPT                 = 0000001E
DYN$C_IRP                 = 0000000A
DYN$C_ORB                 = 00000049
DYN$C_TQE                 = 0000000F
DYN$C_UCB                 = 00000010
EXE$ABORTIO                 ********    X  03
EXE$ALLOCBUF                ********    X  03
EXE$ALONONPAGED             ********    X  03
EXE$ALOPHYCNTG              ********    X  03
EXE$BUFFRQUOTA              ********    X  03
EXE$BUFQUOPRC               ********    X  03
EXE$DEANONPAGED             ********    X  03
EXE$FINISHIO                ********    X  03
EXE$FORK                    ********    X  03
EXE$GB_CPUTYPE              ********    X  03
EXE$GL_TENUSEC              ********    X  03
EXE$GL_UBDELAY              ********    X  03
EXE$GQ_SYSTIME              ********    X  03
EXE$INSTIMQ                 ********    X  03
EXE$PROBER_DSC              ********    X  03
EXE$QIORETURN               ********    X  03
EXE$READCHK                 ********    X  03
EXE$WRITECHK                ********    X  03
FFI$L_DL_UCB              = 00000034
FFI$L_ERROR               = 0000001C
FFI$L_RECV_DONE           = 00000018
FFI$L_SHUT_DONE           = 00000020
FFI$L_XMIT                = 00000010
FFI$L_XMIT_DONE           = 00000014
FFI_INIT                    00000270 RG    03
FILCRCVLIST                 0000143E RG    03
FIND_MLTENTRY               00002B50 R     03
FIND_POINT_UCB              000025F7 R     03
FIND_SHR                    000025BB R     03
FINISH_RCV_FFI              00001AD8 RG    03
FINISH_RCV_IO               00001B13 RG    03
FINISH_XMT_FFI              00001ACA RG    03
FKB$B_FIPL                = 0000000B
FKB$B_TYPE                = 0000000A
FKB$L_FR3                 = 00000010
FORK_PROC                   000016E0 RG    03
FORK_TIMER                  0000132C R     03
FUNCTAB_LEN               = 00000040
GET_CHAR_BUF                00000D90 R     03
IDB$L_CSR                 = 00000000
IDB$L_UCBLST              = 00000018
INACT_ERROR                 000006BC R     03
INIT_C_AQUOTA             = 00000002
INIT_C_BUFSIZE            = 00000080
INIT_C_QUOTA              = 00002328
INTEXIT                     000016B7 R     03
IO$V_ATTNAST              = 00000008
IO$V_CLR_COUNT            = 0000000A
IO$V_CTRL                 = 00000009
IO$V_NOW                  = 00000006
IO$V_RD_COUNT             = 00000008
IO$V_SHUTDOWN             = 00000007
IO$V_STARTUP              = 00000006
IO$_READLBLK              = 00000021
IO$_READPBLK              = 0000000C
IO$_READVBLK              = 00000031
IO$_SENSECHAR             = 0000001B
IO$_SENSEMODE             = 00000027
IO$_SETCHAR               = 0000001A
IO$_SETMODE               = 00000023
IO$_VIRTUAL               = 0000003F
IO$_WRITELBLK             = 00000020
IO$_WRITEPBLK             = 0000000B
IO$_WRITEVBLK             = 00000030
IOC$ALOUBAMAP               ********    X  03
IOC$CREDIT_UCB              ********    X  03
IOC$DELETE_UCB              ********    X  03
IOC$INITIATE                ********    X  03
IOC$LOADUBAMAP              ********    X  03
IOC$LOADUBAMAPA             ********    X  03
IOC$LOADUBAMAPN             ********    X  03
```

```
IOC$MNTVER                  ********   X   03
IOC$PURGDATAP               ********   X   03
IOC$RELDATAP                ********   X   03
IOC$RELMAPREG               ********   X   03
IOC$RETURN                  ********   X   03
IO_DONE                     00001BDC R     03
IO_DONE1                    00001BE0 R     03
IO_DONE2                    00001BF9 R     03
IO_DONE3                    00001BC5 R     03
IPL$_ASTDEL               = 00000002
IPL$_QUEUEAST             = 00000006
IPL$_SYNCH                = 00000008
IPL$_TIMER                = 00000008
IPL$_XQ_DIPL              = 00000015
IPL$_XQ_FIPL              = 00000008
IRP$B_EFN                 = 00000022
IRP$B_PRI                 = 00000023
IRP$B_RMOD                = 0000000B
IRP$B_TYPE                = 0000000A
IRP$B_XQ_DATAP              00000046   G
IRP$B_XQ_FUNC               00000021   G
IRP$B_XQ_RING               00000039   G
IRP$B_XQ_SLOT               00000038   G
IRP$C_LENGTH              = 000000C4
IRP$C_XQ_STD                00000060   G
IRP$L_AST                 = 00000010
IRP$L_ASTPRM              = 00000014
IRP$L_BCNT                = 00000032
IRP$L_DIAGBUF             = 0000004C
IRP$L_FQFL                = 00000060
IRP$L_IOSB                = 00000024
IRP$L_IOST1               = 00000038
IRP$L_IOST2               = 0000003C
IRP$L_LBOFF               = 00000090
IRP$L_MEDIA               = 00000038
IRP$L_PID                 = 0000000C
IRP$L_RBOFF               = 00000098
IRP$L_RBUFH_AD            = 00000094
IRP$L_SVAPTE              = 0000002C
IRP$L_UCB                 = 0000001C
IRP$L_WIND                = 00000018
IRP$L_XQ_DATBUF             0000003C   G
IRP$L_XQ_DGUNI              00000040   G
IRP$L_XQ_MAP                0000003C   G
IRP$L_XQ_P2BUF              00000040   G
IRP$L_XQ_SETUP              00000094   G
IRP$L_XQ_SHR                00000090   G
IRP$L_XQ_SYSBUF             0000003C   G
IRP$L_XQ_UPADR              00000044   G
IRP$L_XQ_USERBUF            0000003C   G
IRP$M_CHAINED             = 00000020
IRP$M_COMPLX              = 00000008
IRP$M_DIAGBUF             = 00000080
IRP$Q_STATION             = 00000040
IRP$V_CHAINED             = 00000005
IRP$V_DIAGBUF             = 00000007
IRP$V_FUNC                = 00000001
IRP$W_BCNT                = 00000032
IRP$W_BOFF                = 00000030
IRP$W_CHAN                = 00000028
IRP$W_FUNC                = 00000020
IRP$W_SIZE                = 00000008
IRP$W_STS                 = 0000002A
IRP$W_XQ_CODE               00000040   G
IRP$W_XQ_P2SIZ              00000044   G
IRP$W_XQ_PROTYP             0000003A   G
IRP$W_XQ_RID                00000038   G
IRP$W_XQ_STATUS             0000003A   G
IRP$W_XQ_USERSIZ            00000038   G
JIB$L_BYTCNT              = 00000020
JIB$L_BYTLM               = 00000024
LINE_CTR                    00000109 R     03
LINE_CTR_BUFSIZ           = 0000006A
LINE_CTR_SIZE             = 00000014
LINE_PARAM                  00000081 R     03
LINE_PARAM_WO               00000078 R     03
LINE_PRM_BUFSIZ           = 0000007A
LOAD_PORT                   000015FE RG    03
MASKH                     = 00000080
MASKL                     = 08000000
MATCH_ADDRESS               00002C69 R     03
MATCH_MULTI                 00002C3A R     03
MATCH_PROMTYP               00002CE8 R     03
MATCH_PROTYP                00002CC6 R     03
MATCH_SHR                   0000070B R     03
MATCH_SRC                   00001A12 R     03
MAX_BUFSIZ_UV1            = 000005EA
MAX_C_CHAIN               = 00000001
MAX_C_MLT                 = 0000000C
MAX_C_RCV                 = 00000008
MAX_C_RCVUV1              = 00000005
MAX_C_XMT                 = 00000004
MAX_C_XMTUV1              = 00000001
MAX_PKT_SIZE              = 000005DC
MIN_PKT_SIZE              = 0000002E
MMG$GL_SPTBASE              ********   X   03
MOPCTRTAB                   00000171 R     03
MOP_CTR_BUILD               00001DC4 R     03
MOP_CTR_REQUEST             00001D8F RG    03
MOP_CTR_SIZE              = 00000051
MOVE_MULTI                  00002728 R     03
NEXTMSG                     00001CCC R     03
NI_CTR_PROTYP             = 00000260
NI_CTR_READ               = 00000009
NI_CTR_REPLY              = 0000000B
NMA$C_ACC_EXC             = 00000003
NMA$C_ACC_LIM             = 00000002
NMA$C_ACC_SHR             = 00000001
NMA$C_CTCIR_BRC           = 000003E8
NMA$C_CTCIR_BSN           = 000003E9
NMA$C_CTCIR_DBR           = 000003F2
NMA$C_CTCIR_DBS           = 000003F3
NMA$C_CTCIR_MNE           = 00000A8D
NMA$C_CTCIR_UBU           = 00000429
```

```
NMA$C_CTLIN_BID         = 000003F5
NMA$C_CTLIN_BRC         = 000003E8
NMA$C_CTLIN_BS1         = 000003F6
NMA$C_CTLIN_BSM         = 000003F7
NMA$C_CTLIN_BSN         = 000003E9
NMA$C_CTLIN_CDC         = 00000425
NMA$C_CTLIN_DBR         = 000003F2
NMA$C_CTLIN_DBS         = 000003F3
NMA$C_CTLIN_LBE         = 00000411
NMA$C_CTLIN_MBL         = 000003F4
NMA$C_CTLIN_MBS         = 00000A8D
NMA$C_CTLIN_MBY         = 000003EA
NMA$C_CTLIN_MSN         = 00000A8E
NMA$C_CTLIN_OVR         = 00000428
NMA$C_CTLIN_RFL         = 00000426
NMA$C_CTLIN_SBU         = 00000429
NMA$C_CTLIN_SFL         = 00000424
NMA$C_CTLIN_UBU         = 0000042A
NMA$C_CTLIN_UFD         = 00000427
NMA$C_CTLIN_ZER         = 00000000
NMA$C_LINCN_LOO         = 00000001
NMA$C_LINCN_NOR         = 00000000
NMA$C_LINMC_CAL         = 00000003
NMA$C_LINMC_CLR         = 00000002
NMA$C_LINMC_SDF         = 00000004
NMA$C_LINMC_SET         = 00000001
NMA$C_LINPR_NI          = 00000006
NMA$C_LINPR_POI         = 00000000
NMA$C_PCCI_MST          = 00000AFA
NMA$C_PCLI_ACC          = 00000B1E
NMA$C_PCLI_BFN          = 00000451
NMA$C_PCLI_BSZ          = 00000B20
NMA$C_PCLI_BUS          = 00000AF1
NMA$C_PCLI_CON          = 00000456
NMA$C_PCLI_CRC          = 00000B1C
NMA$C_PCLI_DCH          = 00000B1B
NMA$C_PCLI_DES          = 00000B21
NMA$C_PCLI_HBQ          = 00000B1D
NMA$C_PCLI_HWA          = 00000488
NMA$C_PCLI_MCA          = 00000B0F
NMA$C_PCLI_MLT          = 00000B19
NMA$C_PCLI_PAD          = 00000B1A
NMA$C_PCLI_PHA          = 00000B04
NMA$C_PCLI_PRM          = 00000B18
NMA$C_PCLI_PRO          = 00000458
NMA$C_PCLI_PTY          = 00000B0E
NMA$C_STATE_OFF         = 00000001
NMA$C_STATE_ON          = 00000000
NMA$M_CNT_COU           = 00008000
NMA$M_CNT_MAP           = 00001000
NMA$M_CNT_TYP           = 00000FFF
NMA$V_CNT_MAP           = 0000000C
NMA$V_CNT_WID           = 0000000D
NO_SHR                    0000045C R      03
ORB$B_FLAGS             = 0000000B
ORB$L_OWNER             = 00000000
ORB$M_PROT_16           = 00000001
ORB$W_PROT              = 00000018
P1                      = 00000000
P2                      = 00000004
P2B_B_SPARE               0000000B
P2B_B_TYPE                0000000A
P2B_C_LENGTH              0000000C
P2B_L_BUFFER              00000004
P2B_L_POINTER             00000000
P2B_T_DATA                0000000C
P2B_W_SIZE                00000008
P3                      = 00000008
P4                      = 0000000C
P5                      = 00000010
PCB$L_JIB               = 00000080
PCB$L_PID               = 00000060
PCB$L_STS               = 00000024
PCB$Q_PRIV              = 00000084
PCB$V_SSRWAIT           = 0000000A
PHYADD0                   00000000  G
PHYADD1                   00000002  G
PHYADD2                   00000004  G
PHYADD3                   00000006  G
PHYADD4                   00000008  G
PHYADD5                   0000000A  G
POINT                   = 00000001
POKE_USER                 00002C91 R      03
PR$_IPL                 = 00000012
PR$_SID_TYP730          = 00000003
PR$_SID_TYP750          = 00000002
PR$_SID_TYP780          = 00000001
PR$_SID_TYP790          = 00000004
PR$_SID_TYPUV1          = 00000007
PRM_B_FLAG                00000002
PRM_FLG_M_CHECK         = 00000010
PRM_FLG_M_INVALID       = 00000004
PRM_FLG_M_MAX           = 00000002
PRM_FLG_M_MIN           = 00000001
PRM_FLG_V_CDB           = 00000003
PRM_FLG_V_INVALID       = 00000002
PRM_FLG_V_MAX           = 00000001
PRM_FLG_V_MIN           = 00000000
PRM_OFF_M_VALUE         = 000003FF
PRM_OFF_M_WIDTH         = 0000FC00
PRM_OFF_S_VALUE         = 0000000A
PRM_OFF_S_WIDTH         = 00000006
PRM_OFF_V_VALUE         = 00000000
PRM_OFF_V_WIDTH         = 0000000A
PRM_TYP_M_CODE          = 00000FFF
PRM_TYP_M_STRING        = 00001000
PRM_TYP_V_STRING        = 0000000C
PRM_W_OFF                 00000003
PRM_W_TYPE                00000000
PRV$V_PHY_IO            = 00000016
PTE$S_PFN               = 00000015
QNA_INTR                  00001678 RG     03
QUEPKT                    00000CC3 R      03
RCVLIST                   00000004  G
```

```
RCVLST1                     00000006  G
RCV_C_LENGTH                0000000C  G
RCV_DSC_M_CHAIN           = 00004000  G
RCV_DSC_M_VALID           = 00008000  G
RCV_DSC_S_CHAIN           = 00000001  G
RCV_DSC_S_VALID           = 00000001  G
RCV_DSC_V_CHAIN           = 0000000E  G
RCV_DSC_V_VALID           = 0000000F  G
RCV_ERROR                   00001997  R     03
RCV_FDT                     00000651  RG    03
RCV_FLG_M_ERR             = 00004000  G
RCV_FLG_M_LAST            = 00008000  G
RCV_FLG_V_ERR             = 0000000E  G
RCV_FLG_V_LAST            = 0000000F  G
RCV_K_LENGTH              = 0000006C
RCV_START                   000006C2  RG    03
RCV_STS_M_CRCERR          = 00000002  G
RCV_STS_M_DISCARD         = 00001000  G
RCV_STS_M_ERR             = 00004000  G
RCV_STS_M_ESETUP          = 00002000  G
RCV_STS_M_FRAME           = 00000004  G
RCV_STS_M_LAST            = 00008000  G
RCV_STS_M_OVF             = 00000001  G
RCV_STS_M_RLEN            = 00000700  G
RCV_STS_M_RUNT            = 00000800  G
RCV_STS_M_SHORT           = 00000008  G
RCV_STS_S_RLEN            = 00000003  G
RCV_STS_V_CRCERR          = 00000001  G
RCV_STS_V_DISCARD         = 0000000C  G
RCV_STS_V_ERR             = 0000000E  G
RCV_STS_V_ESETUP          = 0000000D  G
RCV_STS_V_FRAME           = 00000002  G
RCV_STS_V_LAST            = 0000000F  G
RCV_STS_V_OVF             = 00000000  G
RCV_STS_V_RLEN            = 00000008  G
RCV_STS_V_RUNT            = 0000000B  G
RCV_STS_V_SHORT           = 00000003  G
RCV_W_ADDR                  00000004  G
RCV_W_ADDRHI                00000002  G
RCV_W_FLAG                  00000000  G
RCV_W_LEN                   00000006  G
RCV_W_LENB                  0000000A  G
RCV_W_STS                   00000008  G
READ_CIRC_CTR               00000CFC  R     03
READ_LINE_CTR               00000CDA  R     03
REG_DUMP                    00001E3E  RG    03
RESTART_DELTA             = 01C9C380
RESTART_ROUT                00001E5A  RG    03
RES_MULTI                   00002754  R     03
RETURN_IRP                  00001EB3  R     03
RETURN_MULTI                00002C06  R     03
RETURN_P2                   00002998  RG    03
RHDR_B_SPARE                0000000B
RHDR_B_TYPE                 0000000A
RHDR_C_DATA               = 0000000E
RHDR_C_LENGTH               0000001A
RHDR_G_DEST                 0000000C
RHDR_G_SRC                  00000012
RHDR_L_BUFFER               00000004
RHDR_L_DATA                 00000000
RHDR_T_DATA                 0000000C
RHDR_W_SIZE                 00000008
RHDR_W_TYPE                 00000018
SAV_MULTI                   0000273E  R     03
SCH$GL_PCBVEC               ********    X   03
SCH$IOLOCKW                 ********    X   03
SCH$IOUNLOCK                ********    X   03
SCHED_FORK                  000016C3  RG    03
SCHED_FORKC                 000016C1  RG    03
SCS$GB_SYSTEMID             ********    X   03
SENSEMODE_FDT               00000C0B  RG    03
SETMODE_FDT                 0000079C  RG    03
SETUP_ERR                   00001369  R     03
SETUP_MODE                  0000136F  R     03
SET_DESAD                   00002BB2  R     03
SET_MULTI                   00002ADA  R     03
SET_MULTIN                  00000BED  R     03
SET_PHYAD                   00002BA7  R     03
SHR_B_STS                   0000000B  G
SHR_B_TYPE                  0000000A  G
SHR_C_LENGTH                0000002A  G
SHR_C_QUEUES              = 00000002
SHR_DEF                     0000043B  R     03
SHR_G_DEST                  00000012  G
SHR_L_PID                   0000000C  G
SHR_L_QBL                   00000004  G
SHR_L_QFL                   00000000  G
SHR_Q_QUEUES                00000018  G
SHR_Q_RCVMSG                00000018  G
SHR_Q_RCVREQ                00000020  G
SHR_STS_M_INITED          = 00000001  G
SHR_STS_V_INITED          = 00000000  G
SHR_UCB                     00000A78  RG    03
SHR_W_CHAN                  00000010  G
SHR_W_QUOTA                 00000028  G
SHR_W_SIZE                  00000008  G
SHUT                        00000E90  R     03
SHUTDOWN                    0000210A  RG    03
SHUTDOWN_PROTYP             000020EB  RG    03
SHUTDOWN_QNA                00001FD6  RG    03
SIZ...                    = 00000006
SQUEEZ_MULTI                00002B74  R     03
SS$_ABORT                 = 0000002C
SS$_ACCVIO                = 0000000C
SS$_BADPARAM              = 00000014
SS$_BUFFEROVF             = 00000601
SS$_COMMHARD              = 000020C4
SS$_CTRLERR               = 00000054
SS$_DATACHECK             = 0000005C
SS$_DATAOVERUN            = 00000838
SS$_DEVALLOC              = 00000840
SS$_DEVINACT              = 000020D4
SS$_DEVOFFLINE            = 00000084
SS$_DEVREQERR             = 00000334
```

```
SS$_DISCONNECT          =  0000204C
SS$_DUPUNIT             =  000021C4
SS$_ENDOFFILE           =  00000870
SS$_EXQUOTA             =  0000001C
SS$_INSFMAPREG          =  00000344
SS$_INSFMEM             =  00000124
SS$_IVBUFLEN            =  0000034C
SS$_NOPRIV              =  00000024
SS$_NORMAL              =  00000001
SS$_OPINCOMPL           =  000002D4
SS$_POWERFAIL           =  00000364
SS$_TIMEOUT             =  0000022C
START                      00000ECE  RG      03
STARTIO                    00000E5E  RG      03
STARTUP                    00000E80  R       03
START_RECEIVE              000014BA  RG      03
START_TIMER                0000133D  R       03
STOP                       00000EB4  R       03
STOP_TQE                   00001F34  R       03
TAKE_QUOTA                 000009E6  R       03
TIMEOUT                    00001F46  RG      03
TIMOUT                     00001F29  R       03
TOSAMSG                    00001CA3  R       03
TOSSMSG                    00001CA6  R       03
TQE$B_RQTYPE            =  0000000B
TQE$B_TYPE              =  0000000A
TQE$C_LENGTH            =  00000030
TQE$C_SSREPT            =  00000005
TQE$C_SSSNGL            =  00000001
TQE$L_FPC               =  0000000C
TQE$L_RQPID             =  0000002C
TQE$M_REPEAT            =  00000004
TQE$Q_DELTA             =  00000020
TQE$W_SIZE              =  00000008
TQE_C_DELTA             =  00000002
TQE_DELTA               =  01312D00
TQE_TIMER                  00001EBD  RG      03
UCB$B_DEVCLASS          =  00000040
UCB$B_DEVTYPE           =  00000041
UCB$B_DIPL              =  0000005E
UCB$B_FIPL              =  0000000B
UCB$B_XQ_ACC               000000D4  G
UCB$B_XQ_BFN               000000D5  G
UCB$B_XQ_CDBPRM            000000DD  G
UCB$B_XQ_CON               000000DD  G
UCB$B_XQ_DCH               000000DC  G
UCB$B_XQ_MLT               000000DB  G
UCB$B_XQ_MLTTBL            000000E6  G
UCB$B_XQ_MST               000000E4  G
UCB$B_XQ_MULTI             000000E5  G
UCB$B_XQ_PAD               000000D9  G
UCB$B_XQ_PRM               000000DA  G
UCB$B_XQ_PRO               000000D8  G
UCB$B_XQ_SETPRM            000000CC  G
UCB$B_XQ_SHRPRM            000000D6  G
UCB$C_NI_LENGTH         =  00000098
UCB$C_XQ_CDBPRM         =  00000001
UCB$C_XQ_LENGTH            00000195  G
UCB$C_XQ_QUEUES         =  00000004
UCB$C_XQ_SETPRM         =  00000007
UCB$C_XQ_SHRPRM         =  00000008
UCB$G_XQ_DES               000000CC  G
UCB$G_XQ_MLTTBL            0000012F  G
UCB$G_XQ_MULTI             000000E7  G
UCB$G_XQ_PHA               000000DE  G
UCB$L_CPID              =  00000020
UCB$L_CRB               =  00000024
UCB$L_DDB               =  00000028
UCB$L_DEVCHAR           =  00000038
UCB$L_DEVDEPEND         =  00000044
UCB$L_LINK              =  00000030
UCB$L_NI_HWAPTR         =  00000090
UCB$L_NI_MLTPTR         =  00000094
UCB$L_ORB               =  0000001C
UCB$L_PID               =  0000002C
UCB$L_SVAPTE            =  00000078
UCB$L_XQ_AST               000000C0  G
UCB$L_XQ_CPID              000000BC  G
UCB$L_XQ_DEFUSR            000000C4  G
UCB$L_XQ_FFI               0000018D  G
UCB$L_XQ_PID               000000B8  G
UCB$L_XQ_RBLCTR            00000183  G
UCB$L_XQ_RBYCTR            00000187  G
UCB$L_XQ_SBLCTR            0000017B  G
UCB$L_XQ_SBYCTR            0000017F  G
UCB$L_XQ_STIRP             00000191  G
UCB$M_INT               =  00000002
UCB$M_ONLINE            =  00000010
UCB$M_POWER             =  00000020
UCB$M_TEMPLATE          =  00002000
UCB$M_TIM               =  00000001
UCB$M_XQ_INITED         =  00000001  G
UCB$M_XQ_INTERLOCK      =  00004000  G
UCB$M_XQ_PROTYP         =  00000004  G
UCB$M_XQ_RESTART        =  00008000  G
UCB$M_XQ_RUN            =  00000010  G
UCB$M_XQ_SHARE          =  00000008  G
UCB$M_XQ_STACK          =  00000040  G
UCB$M_XQ_START          =  00000020  G
UCB$Q_XQ_IOQS              000000A0  G
UCB$Q_XQ_QUEUES            00000098  G
UCB$Q_XQ_RCVMSG            000000A0  G
UCB$Q_XQ_RCVREQ            000000A8  G
UCB$Q_XQ_SHARE             00000098  G
UCB$Q_XQ_XMTREQ            000000B0  G
UCB$V_ONLINE            =  00000004
UCB$V_POWER             =  00000005
UCB$V_XQ_INITED         =  00000000  G
UCB$V_XQ_INTERLOCK      =  0000000E  G
UCB$V_XQ_PROTYP         =  00000002  G
UCB$V_XQ_RESTART        =  0000000F  G
UCB$V_XQ_RUN            =  00000004  G
UCB$V_XQ_SHARE          =  00000003  G
UCB$V_XQ_STACK          =  00000006  G
```

```
UCB$V_XQ_START           =  00000005  G
UCB$W_BCNT               =  0000007E
UCB$W_BOFF               =  0000007C
UCB$W_DEVBUFSIZ          =  00000042
UCB$W_DEVSTS             =  00000068
UCB$W_ERRCNT             =  00000082
UCB$W_REFC               =  0000005C
UCB$W_STS                =  00000064
UCB$W_UNIT               =  00000054
UCB$W_XQ_BSZ                000000D6  G
UCB$W_XQ_CTR                00000177  G
UCB$W_XQ_HBQ                000000D2  G
UCB$W_XQ_MNECTR             00000177  G
UCB$W_XQ_PROTYP             000000CA  G
UCB$W_XQ_QUOTA              000000C8  G
UCB$W_XQ_TOTQUO             0000018B  G
UCB$W_XQ_UBUCTR             00000179  G
UNIT_INIT                   000001BA  RG    03
UV1_BUFFER_AREA          =  0000237C
UV1_BUFFER_LENGTH        =  00002400
UV1_BUFFER_PAGES         =  00000012
VA$M_BYTE                =  000001FF
VA$S_VPN                 =  00000015
VA$V_VPN                 =  00000009
VALIDATE_P2                 0000276A  RG    03
VALID_MULTI                 00002A66  R     03
VALID_PHYAD                 00002AA2  R     03
VEC$B_DATAPATH           =  00000013
VEC$B_NUMREG             =  00000012
VEC$L_IDB                =  00000008
VEC$L_INITIAL            =  0000000C
VEC$L_START              =  0000001C
VEC$L_UNITINIT           =  00000018
VEC$W_MAPREG             =  00000010
VECTOR                      0000000C  G
XBUF_C_HEADER               0000000E
XBUF_G_DEST                 00000000
XBUF_G_SRC                  00000006
XBUF_T_DATA                 0000000E
XBUF_W_SIZE                 0000000E
XBUF_W_TYPE                 0000000C
XM$M_STS_ACTIVE          =  00000800
XM$V_ERR_FATAL           =  00000010
XM$V_ERR_START           =  00000017
XM$V_STS_ACTIVE          =  0000000B
XM$V_STS_BUFFAIL         =  0000000C
XM$V_STS_TIMO            =  00000009
XMTLIST                     00000008  G
XMTLST1                     0000000A  G
XMT_ALT_START               00000493  RG    03
XMT_C_LENGTH                0000000C  G
XMT_C_TIM                =  00000008
XMT_DSC_M_BEGODD         =  00000040  G
XMT_DSC_M_CHAIN          =  00004000  G
XMT_DSC_M_ENDODD         =  00000080  G
XMT_DSC_M_EOM            =  00002000  G
XMT_DSC_M_SETUP          =  00001000  G
XMT_DSC_M_VALID          =  00008000  G
XMT_DSC_S_BEGODD         =  00000001  G
XMT_DSC_S_CHAIN          =  00000001  G
XMT_DSC_S_ENDODD         =  00000001  G
XMT_DSC_S_EOM            =  00000001  G
XMT_DSC_S_SETUP          =  00000001  G
XMT_DSC_S_VALID          =  00000001  G
XMT_DSC_V_BEGODD         =  00000006  G
XMT_DSC_V_CHAIN          =  0000000E  G
XMT_DSC_V_ENDODD         =  00000007  G
XMT_DSC_V_EOM            =  0000000D  G
XMT_DSC_V_SETUP          =  0000000C  G
XMT_DSC_V_VALID          =  0000000F  G
XMT_ERROR                   000019C3  R     03
XMT_FDT                     00000291  RG    03
XMT_FFI_START               0000034C  RG    03
XMT_FLG_M_ERR            =  00004000  G
XMT_FLG_M_LAST           =  00008000  G
XMT_FLG_V_ERR            =  0000000E  G
XMT_FLG_V_LAST           =  0000000F  G
XMT_INITIATE                000003D7  RG    03
XMT_K_LENGTH             =  0000003C
XMT_START                   0000038F  RG    03
XMT_STS_M_COL            =  000000F0  G
XMT_STS_M_ERR            =  00004000  G
XMT_STS_M_LAST           =  00008000  G
XMT_STS_M_LCAR           =  00001000  G
XMT_STS_M_NOCAR          =  00000800  G
XMT_STS_S_COL            =  00000004  G
XMT_STS_V_ABORT          =  00000009  G
XMT_STS_V_COL            =  00000004  G
XMT_STS_V_ERR            =  0000000E  G
XMT_STS_V_FAIL           =  00000008  G
XMT_STS_V_LAST           =  0000000F  G
XMT_STS_V_LCAR           =  0000000C  G
XMT_STS_V_NOCAR          =  0000000B  G
XMT_TDR_M_TDR            =  00003FFF  G
XMT_TDR_S_TDR            =  0000000E  G
XMT_TDR_V_TDR            =  00000000  G
XMT_TIM                  =  00000005
XMT_UV1                     000005C3  R     03
XMT_W_ADDR                  00000004  G
XMT_W_ADDRHI                00000002  G
XMT_W_FLAG                  00000000  G
XMT_W_LEN                   00000006  G
XMT_W_STS                   00000008  G
XMT_W_TDR                   0000000A  G
XQ$DDT                      00000000  RG    03
XQ_CSR_M_CAR             =  00002000  G
XQ_CSR_M_ELOOP           =  00000200  G
XQ_CSR_M_ERR             =  00004000  G
XQ_CSR_M_ILOOP           =  00000100  G
XQ_CSR_M_INTENA          =  00000040  G
XQ_CSR_M_NXM             =  00000004  G
XQ_CSR_M_RCVENA          =  00000001  G
XQ_CSR_M_RCVINT          =  00000080  G
XQ_CSR_M_RCVINV          =  00000020  G
```

```
XQ_CSR_M_RESET                  =  00000002  G
XQ_CSR_M_RROM                   =  00000008  G
XQ_CSR_M_SANITY                 =  00000400  G
XQ_CSR_M_XCAB                   =  00001000  G
XQ_CSR_M_XMTINT                 =  00008000  G
XQ_CSR_M_XMTINV                 =  00000010  G
XQ_CSR_V_CAR                    =  0000000D  G
XQ_CSR_V_ELOOP                  =  00000009  G
XQ_CSR_V_ERR                    =  0000000E  G
XQ_CSR_V_ILOOP                  =  00000008  G
XQ_CSR_V_INTENA                 =  00000006  G
XQ_CSR_V_NXM                    =  00000002  G
XQ_CSR_V_RCVENA                 =  00000000  G
XQ_CSR_V_RCVINT                 =  00000007  G
XQ_CSR_V_RCVINV                 =  00000005  G
XQ_CSR_V_RESET                  =  00000001  G
XQ_CSR_V_RROM                   =  00000003  G
XQ_CSR_V_SANITY                 =  0000000A  G
XQ_CSR_V_XCAB                   =  0000000C  G
XQ_CSR_V_XMTINT                 =  0000000F  G
XQ_CSR_V_XMTINV                 =  00000004  G
XQ_C_ADDRCV                     =  00000040
XQ_C_CNTSIZ                     =  00000002
XQ_C_CRC                        =  00000004
XQ_C_HEADER                     =  0000000E
XQ_C_STPRO                      =  00000660
XQ_END                             00002CF5 RG    03
XQ_FC_V_CANCEL                  =  00000004  G
XQ_FC_V_CHMODE                  =  00000006  G
XQ_FC_V_INIT                    =  00000000  G
XQ_FC_V_RECV                    =  00000002  G
XQ_FC_V_RESTART                 =  00000005  G
XQ_FC_V_STOP                    =  00000003  G
XQ_FC_V_XMIT                    =  00000001  G
XQ_FUNCTABLE                       00000038 R     03
XQ_SOFT_M_POWER                 =  00000002  G
XQ_SOFT_M_TIMEOUT               =  00000001  G
XQ_SOFT_V_POWER                 =  00000001  G
XQ_SOFT_V_TIMEOUT               =  00000000  G
_$$_                            =  00000000
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
ABS	000002F4 (756.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
$$$105_PROLOGUE	000000B9 (185.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE
$$$115_DRIVER	00002CF5 (11509.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	36	00:00:00.07	00:00:01.40
Command processing	138	00:00:00.46	00:00:04.32
Pass 1	1819	00:00:55.23	00:03:47.44
Symbol table sort	0	00:00:04.90	00:00:20.55
Pass 2	478	00:00:14.82	00:00:55.84
Symbol table output	3	00:00:00.49	00:00:01.36
Psect synopsis output	2	00:00:00.01	00:00:00.01
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	2479	00:01:15.98	00:05:10.92

The working set limit was 3600 pages.
436246 bytes (853 pages) of virtual memory were used to buffer the intermediate code.
There were 250 pages of symbol table space allocated to hold 4250 non-local and 656 local symbols.
8230 source lines were read in Pass 1, producing 65 object records in Pass 2.
103 pages of virtual memory were used to define 92 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SHRLIB]NMALIBRY.MLB;1	1
_255DUA28:[SYS.OBJ]LIB.MLB;1	42
_255DUA28:[SYSLIB]STARLET.MLB;2	14
TOTALS (all libraries)	57

4109 GETS were required to define 57 macros.

There were no errors, warnings or information messages.

```
MACRO/LIS=LIS$:XQDRIVER/OBJ=OBJ$:XQDRIVER MSRC$:XQDRIVER/UPDATE=(ENH$:XQDRIVER)+EXECML$/LIB+SHRLIB$:NMALIBRY/LIB
```

XQDRIVER
LIS

XMDRIVER
LIS

XWDRIVER
LIS

DTGLOBAL
LIS

DTDEFINE
LIS

DTRMAIN
LIS

DTRAST
LIS

DTPREFIX
MAR

DTSDTR

DTCOMMON
LIS

DTRECV
MAP

DTSEND
MAP

DTMACROS
MAR